/*REVER O TITULO*/
The use of Evolving Graph Combinatorial Model in Routing Protocols for Dynamic Networks

Julian Monteiro - Advisor: Alfredo Goldman

Instituto de Matemática e Estatística – Universidade de São Paulo (IME-USP)
Rua do Matão, 1010, São Paulo, SP, 05508-090 – Brazil
{jm,gold}@ime.usp.br

Abstract. The assessment of routing protocols for ad hoc networks is a difficult task, due to the networks’ highly dynamic behavior and the absence of benchmarks. Recently, a graph theoretic model – the evolving graphs – was proposed to help capture the network topology changes during time, with predictable dynamics at least. The algorithms and insights obtained through this model are theoretically very efficient and intriguing. However, there is no study about the use of such theoretical results into practical situations. We used the NS2 network simulator to first implement an evolving graph based routing protocol, and then used it as a benchmark when comparing four major ad-hoc routing protocols. Interestingly, our experiments showed that evolving graphs have the potential to be an effective and powerful tool. In order to make this model widely applicable, however, some practical issues still have to be addressed and incorporated into the model.

1 Introduction and Motivation

Wireless communication networks have become increasingly popular in the computing industry and are widely available in our every day life. A Mobile Ad hoc NETwork (MANET) is a collection of mobile devices that are dynamically connected in an arbitrary manner, without the aid of any established infrastructure or centralized administration [6, 23]. These mobile devices with wireless transmitters are called nodes. When two nodes want to communicate, they may not be within each other’s range, but they may communicate if other nodes between them also participate in the network, acting as routers, forwarding packets to the other end. These are called multi-hop wireless ad hoc networks.

/*JM: AINDA NAO ADICIONEI UM PARAGRAFO COM 'IMPORTANCIA DAS MANETS'*/

In several environments these nodes are free to move and they may have nonuniform characteristics, driving the emergence of complex ad hoc networks that may have a highly dynamic behavior. Thus, a large number of routing protocols have been developed for MANETs [1,12,15]. Besides the mobility, such protocols must deal with the typical limitations of these networks, like energy limitations, low processing capacity, low bandwidth, and high error rates.
There are different approaches which try to optimize the cost of a routing path, but, until recently, most of them did not take into account the fact that some MANETs may have known connectivity patterns. This happens on some special cases on DTNs [12, 26], such as LEO satellite [4, 8] and Wireless Sensor Networks (WSNs), where, due to energy limitations, network nodes can be scheduled to sleep in given periods [1, 14, 22, 24]. In this kind of networks, henceforth referred to as *Fixed Schedule Dynamic Networks* (FSDNs) [7], the topology dynamics at different time intervals can be predicted (see Fig. 1). Therefore, the performance evaluation of routing protocols should be easier, although a formal tool for benchmarking such protocols is still to become a standard.

/*JM: troquei a fig.1 de lugar*/

![Fig. 1](image_url)

*Fig. 1.* The evolution of a MANET over time. The indices correspond to successive snapshots in time. “Zzz” indicates a sleeping node.

### 1.1 Our Contribution

In this work, we used *evolving graphs* (EG) – a formal abstraction for dynamic networks [3,7] –, in order to design and evaluate least cost routing protocols for MANETs with known connectivity patterns. These protocols are then used as benchmarks for establishing fair comparisons between the four MANET routing protocols, namely DSDV [20], DSR [13], AODV [21] and OLSR [11]. This is done through extensive simulations using *NS2* network simulator [19] within different realistic scenarios.

The concept of *EG* and some least cost journeys were detailed in past articles, however, to the best of our knowledge, none experimental simulations was done until now (besides the publications related to this work [9,17,18]).

We note that the algorithms and insights previously obtained through the EG model are theoretically very efficient and intriguing. The central objective
of our work is thus to assess the usages of these theoretical results in practical situations, where packet dropping, for instance, may pose unexpected challenges to the EG algorithms.


/*JM: NOVA SUBSECAO*/

1.2 Paper Organization

The remainder of this work is organized as follows: In the next section we describe the concept of evolving graphs. The routing protocols to be compared are defined in Section 3. Section 4 shows the simulation environment and the implementation choices. Section 5 presents a selected set of the complete simulation results and analyses. We close this work with our conclusions and avenues for future research in Section 6.

2 Evolving Graph Model

The evolving graph model, proposed in [7], aims to represent a formal abstraction of dynamic networks, through the formalisation of a time domain in graphs.

As an example, consider the four snapshots taken at different time intervals of a MANET, as depicted in Fig. 1. As one can readily observe, nodes $D$ and $G$ are never connected on a single time interval. Notwithstanding, $D$ can indeed send messages to $G$, using the path over time composed of $D, C, E, F, G$. Surprisingly, this otherwise trivial fact cannot be directly modeled by usual graphs.

Concisely, an evolving graph is an indexed sequence of $\tau$ subgraphs of a given graph, where the subgraph at a given index corresponds to the network connectivity at the time interval indicated by the index number, as shown in Fig. 2.
The time domain is further incorporated into the model by restricting journeys (i.e., the equivalent of paths over time) to never move into edges which only existed in past subgraphs. A journey in an evolving graph is thus a path in the underlying graph whose edge time-labels are in a non-decreasing order. Now, it is easy to see in Fig. 2 that \( D, C, E, F, G \) is a journey, as mentioned above. Further, note that \( D, C, E, G \) is also a journey, with less hops, but delivering the message later (in time interval 3 instead of 2), giving rise to different objective functions that may be optimized.

### 2.1 Journey Metrics

In the pursuit of an optimal journey in networks with known connectivity patterns, three metrics have been formalized until now for EGs [3]. They are the foremost, shortest, and fastest journey, which find, respectively, the earliest arrival date, the minimum number of hops, and the minimum delay (time span) route. These three parameters can be individually optimized in polynomial time [3].

/*JM: Alterei o paragrafo seguinte e adicionei 2 subsecoes com os algoritmos*/ We implemented in this work the foremost and shortest journeys algorithms, which are detailed in the next sections. The routing protocol originated by these algorithms will be henceforth referred to as \( EG_{\text{Foremost}} \) and \( EG_{\text{Shortest}} \), respectively.

### 2.2 Foremost Journey Algorithm

Remind that, in order to compute shortest paths, the usual Dijkstra’s algorithm [5] proceeds by building a set \( C \) of closed vertices, for which the shortest paths have already been computed, then choosing a vertex \( u \) not in \( C \) whose shortest path estimate, \( d(u) \), is minimum, and adding \( u \) to \( C \), i.e., closing \( u \). At this point, all arcs from \( u \) to \( V - C \) are opened, i.e., they are examined and the respective shortest path estimate, \( d \), is updated for all end-points. In order to have quick access to the best shortest path estimate, the algorithm keeps a min-heap priority queue \( Q \) with all vertices in \( V - C \), with key \( d \). Note that \( d \) is initialized with \( \infty \) for all vertices but for \( s \), which has \( d = 0 \) (in terms of routing protocols, \( d \) needs to be initialized with the current time \( t \)).

The main observation in Dijkstra’s method is that prefix paths of shortest paths are shortest paths themselves. Unfortunately, the prefix journey of a foremost journey is not necessarily a foremost journey (e.g., considering the \( EG \) in Fig. 2, a message sent at time interval 1 from \( A \) to \( G \) can use the journey \( A, B, E, G \), with the packet reaching \( G \) at time interval 3. The prefix journey \( A, B, E \) will reach \( E \) at time interval 2, although this is not a foremost journey from \( A \) to \( E \), which is in fact \( A, B, C, E \), arriving at moment 1). On the other hand, it was proven that there exists at least one foremost journey with such a property in an evolving graph [3,7].

To compute the foremost journey starting at time \( t \) from \( s \) to all other nodes, we use a direct adaptation of Dijkstra, sketched below, as detailed in [3]:
**Input:** An evolving graph $G$, a vertex $s \in V_G$ and current time $t_{now}$.

**Output:** Tree $T$, which gives the earliest arrival date from $s$.

**Variables:** A min-heap $Q$ of vertices and a vector $d_G$ containing vertices costs (i.e. times).

1. Set $d(s) = t_{now}$, and $d(u) = \infty$ for all other nodes.
2. Initialize min-heap $Q$, sorted by $d$, with only $s$ in the root.
3. While $Q \neq \emptyset$ do
   (a) $x \leftarrow$ root of heap $Q$.
   (b) Delete the root of heap $Q$.
   (c) For each open neighbor $v$ of $x$ do
      i. Compute first valid edge schedule time greater or equal to current time step
      ii. Insert $v$ in the heap $Q$ if it was not there already.
      iii. If needed, update $d(v)$ and its key.
   (d) Update the heap $Q$.
   (e) Close $x$. Insert it in the foremost journeys tree $T$.
4. Return the Tree $T$.

At the end, we have a tree $T$ with the Foremost journey paths from $s$ (starting at time $t_{now}$) to all other nodes. Note that the computation of the first valid edge schedule done at the inner loop may take into account the traversal time for the edge, i.e., the duration of the transmission, if needed. This is the case of timed evolving graph [7].

A foremost journey from a source node $s$ to all other nodes can thus be computed in $O(M(\log \delta_E + \log N))$ time, where $N$ is the number of vertices, $M$ is the number of edges and $\delta_E$ is the maximum number of presence time intervals over all edges. The term $\log \delta_E$ stems from the lookups into the schedule list of intervals, which is required to decide the earliest time interval in which to cross each visited edge.

### 2.3 Shortest Journey Algorithm

The algorithm to compute the Shortest Journey starting at time $t$ from $s$ to all other nodes is sketched below, and detailed in [3]:

**Input:** An evolving graph $G$, a vertex $s \in V_G$ and current time $t_{now}$.

**Output:** The shortest path tree $T$, and an array $\text{location} : V_G \to T$.

**Variables:** The tree $T$ of pairs $(u, t) \in V_G \times \mathbb{R}_+^*$, an integer $d$, an array $\text{earliest} : V_G \to \mathbb{R}_+^*$.

1. Initialize $T = (s, t_{now})$.
2. $\text{earliest}(s) = t_{now}$, and $\text{earliest}(u) = \infty$ for others.
3. $d = 1$.
4. $\text{location}(s) = (s, t_{now})$.
5. While there is $u \in V_G$ such that $\text{location}(u)$ is not defined, do:
(a) For all pairs \((u, t)\) in the tree at depth \(d\), do:
   i. Get all the neighbour pairs \((v, t')\) of \((u, t)\).
   ii. If \(\text{location}(v)\) is not defined, then \(\text{location}(v) = (v, t')\).
   iii. If \(\text{earliest}(v) > t'\), then \(\text{earliest}(v) = t'\) and \((v, t')\) is a son of \((u, t)\) in \(T\).
(b) \(d = d + 1\).
6. return tree \(T\) and the array \(\text{location}\).

In this algorithm, the tree \(T\) represents the shortest journeys from \((s, t_{\text{now}})\) to pairs \((u, t)\), and the array \(\text{location}\) tell where one must look for a vertex \(u\) in the tree to have the shortest journey from \(s\) to \(u\), since a vertex may appear several times. The array \(\text{earliest}\) tells when a vertex \(u\) is reached earliest in the tree. Indeed, if there is a journey taking more than \(k\) hops from \(s\) to \(u\), this journey is relevant if its arrival date is smaller than the former journey’s arrival date.

Once the tree has been properly computed, the retrieval of a shortest journey is straightforward: as the \(\text{location}\) of a vertex \(u\) gives it corresponding pair \((u, t)\) in the tree \(T\), we retrieve the journey by looking for the father of \((u, t)\) in \(T\).

The complexity of the algorithm is \(O(MD)\) [3], where \(M\) is the number of edges and \(D\) is the hop diameter of \(G\).

### 3 Routing Protocols for MANETs

The construction and benchmarking of routing protocols for these spontaneous networks, like the MANETs, could be very challenging. Indeed, a great deal of work has been produced comparing the performance of the four main MANET routing protocols, namely DSR [13], AODV [21], DSDV [20], OLSR [11] that were designed to provide routes in connected networks [2, 6]. The two formers are based on reactive protocols, i.e., the routing path is discovery on-demand as soon a packet must be transmitted. The other two, DSDV and OLSR are proactive protocols as they require periodic exchange of messages to maintain up-to-date their routing tables. We summarize the behavior of these protocols below:

**DSR - Dynamic Source Routing:** reactive protocol, allowing nodes to dynamically discover a route to destination. Such routes are stored in a route cache to enhance the performance. Source routing means that each packet carries in its header the complete ordered list of nodes (the path) to the destination, so that the forwarding nodes do not need to have the routing information. There is a clear compromise between the size of routing tables and packet size.

**AODV - Ad-hoc On-Demand Distance Vector Routing:** reactive protocol, based on DSR and DSDV, which requests a route when needed, and each node maintain a traditional routing table to destinations in use. A routing table entry is \(\text{expired}\) if not used recently.
DSDV - Destination-Sequenced Distance Vector: proactive table-driven protocol based on the distributed Bellman-Ford algorithm, with loop-freedom improvement. Each node has a routing table for all reachable nodes, which stores for each destination the next-hop, the number of hops, and a sequence number. DSDV requires periodical flooding to update the routing table.

OLSR - Optimized Link State Routing: proactive table-driven protocol. Inherits the use of link state algorithm, using shortest path first forwarding. It periodically exchanges the topology information with neighbors, and every node maintains the topology of the whole network. To minimize flooding, OLSR uses nodes that act as Multi Point Relays (MPR). Only these special nodes are responsible for forwarding control traffic. As DSDV, this is a proactive protocol, so the routing paths are available immediately when needed.

There are many other routing protocols with specialized characteristics [15]. We did not evaluate them in this work, mainly because these experiments aimed to compare EG based ones with massively tested and analyzed protocols, as are the four above.

4 Simulation Environment

In order to evaluate the performance analysis of the proposed protocols, we have conducted extensive experiments using the well-known NS2 simulator [19]. NS2 is a discrete event packet level simulator which provides IEEE 802.11 Medium Access Control protocol [10] and realistic radio and physical layer. The modeled radio interface uses an omni-directional antenna with nominal propagation range of 250m.

In the simulations, 50 nodes are randomly placed in a 1500m x 500m field. The simulation time is 900 seconds. A number of 10 constant bit rate (CBR) UDP traffic flows are chosen between node pairs. The average traffic rate is 2 packets/sec, with 256 bytes long packets. The interface queue length at link layer (IFQ) is the default with length 50 packets, however, at some experiments it was changed to 500 packets. These characteristics were chosen to address a sensor network-like environment, where dedicated sensor nodes are constantly collecting data at a low data rate.

4.1 Mobility Models

The dynamic topology and the connectivity pattern of the network are defined by the mobility models. We divided our experiments into two separate kinds of scenarios. One uses the popular random waypoint (RWP) model [2, 13], while the other uses a new mobility model, called intermittent model, which is more suitable to the case of wireless sensor networks and is explained below.

In the Random Waypoint (RWP) mobility model, a mobile node moves to a randomly chosen location, with speed randomly chosen from 1 to 20 m/s, and
pauses for a uniformly chosen time between 0 and \( \text{PauseTime} \). The simulation was run with values of \( \text{PauseTime} \) varying from 0 (continuous motion) to 900 seconds (very low mobility in the network). The use of this classical scenario, yet with its known limitations, is important to compare the results with other performance studies [6].

The **Intermittent Model**, proposed by this work, is based on fixed position nodes that remain uninterruptedly turning themselves on and off (awake and sleep) in given periods (see Fig. 3). Here, the parameters we change are the \( \text{SleepProb} \) (ranging from 0 to 50\%), and the \( \text{HoldTime} \) (ranging from 15s to 180s). In the beginning of the simulation each node is awake, and for the entire simulation it has a \( \text{SleepProb} \) probability to go to sleep (turning itself off). If a node goes to sleep, it remains off for a uniformly randomly chosen \( \text{HoldTime} \). Once this time expires, the node is turned on and stays awake for another period based on \( \text{HoldTime} \). This new model aims to capture the behavior of networks with previously given node activity schedules.

![Lifecycle of a node in the Intermittent Model.](image)

In both models, for each evaluated parameter, we created 20 random scenarios with different random seeds. All scenarios were run on each routing protocol (AODV, DSDV, DSR, OLSR, \( EG_{\text{Foremost}} \), \( EG_{\text{Shortest}} \)) with the same random seeds. In this version, we omitted the error bars on the figures to improve readability.

### 4.2 NS2 Implementation Details

One of the objectives of this work is to investigate the behavior of the \( EG \) foremost and shortest algorithms as a theoretical optimal routing protocols. In this respect, it is important to mention that its implementation as a distributed routing protocol, i.e., with control messages to distribute the \( EG \), keeping it up to date and fail safe mechanisms, is out of the scope of this work. Therefore, we assume that all nodes have the knowledge of the \( EG \), which makes straightforward
the implementation of the protocol. This is still true if the assumption holds for transmitting nodes only.

In order to generate the evolving graph that will be used as input to our protocol, we read in advance the mobility file used as input in NS2. Therefore, we wrote a program (calceg)* that reads a tcl mobility file and captures the node movements to generate a corresponding EG.

Before the simulation begins, this EG of the network is distributed among all nodes. Note that, to calculate the corresponding EG, the transmission and reception range of each node must be taken into account; in our case, it is fixed at exactly 250 meters. We do not use any discretization technique. The computation of the nodes movements to generate the EG are done analytically. This way, each node in the simulated network knows exactly the connectivity pattern of the network during the whole simulation time. When some packet need to be forwarded, the node uses the EG algorithm to calculate the next hop and time to sent the packet.

Each node in the simulated network knows the connectivity pattern of the network during the simulation. This is important for benchmarking purposes, since the EG may be generated and used as a reference when developing or tuning routing protocols and mobility models. Furthermore, there are many practical situations, like those shown in [1,4,16,25,26], in which an EG can be built before the routing phase.

/*JM: O QUE VC QUIS DIZER COM 'RECORDAR QUE ESTE TAMBEM PODE SER O CASO DE DTNs'*/ Each simulation in NS2 generates a trace file, containing all communications that have been done between nodes, including the MAC layer. These files were analyzed to consolidate the results, which are shown in the next section.

5 Simulation Results and Analysis

In this section we show the results obtained by simulation of a network composed of wireless mobile nodes that move around, go to sleep for a while, and communicate with each other. We focused our analysis in several metrics:

- **Average throughput**: The average number of packets received per amount of time (from the first packet sent to the last packet received);
- **Average number of hops**: The average number of hops traversed by the packets;
- **Average end-to-end delay**: The average time between sending and successfully receiving a packet;
- **Energy consumption**: The average energy consumption during transmission and reception phases;
- **Ratio of dropped packets**: Fraction of dropped packets per sent packets.

We also detailed this metric in two others: the number of dropped packets

* The EG implementation and related software can be found at http://julian.com.br/mobidyn/software.
5.1 Foremost Journey and the Average End-to-End delay

The goal of the EG foremost journey algorithm is to calculate journeys that reach the destination as soon as possible. However, in this process some packets may wait for a long time for a connection to be established, and this waiting time is computed in the end-to-end delay metric. In contrast, in the simulation of the other protocols, some of these "late" delivered packets are just being dropped and do not contribute to the end-to-end delay count. In other words, when using the Foremost Journey EG based routing algorithm, the packets end-to-end delay average value is usually larger, even though it was proven in [3] that EGForemost ensures that the packets will reach the destination as soon as possible if a journey exists in the network. This larger values is thus originated by the high delivery rate of the EG protocol, which deliver packets even when the traversal times are longs.

This gives the opportunity to add a parameter Maxdelay in the EG algorithms, which is the maximum delay time that a packet could wait to be delivered in the FSDN. If the calculated delay time is greater than Maxdelay, the packet could be dropped instead of overflowing the network.

Hence, to be fair with the foremost algorithm and better perceive the performance of the EGForemost protocol within others, we calculated the average end-to-end delay taking into account only the packets that have been successfully delivered at the destination for all protocols (i.e. the intersection of received packets). The results in the Fig. 4 shows EGForemost as a lower bound for the end-to-end delay in this scenario. Moreover, almost all packets are delivered.

The proactive protocols, OLSR and DSDV are close to the EGForemost bottom line. The two on top are the reactive ones: AODV and DSR, and thus the higher delay and the non smooth curves are due to the discovery process. The low data rate of our simulations did not help then either, as the route discovery need to be done often and not so much information are sent after that. The results of the EGShortest protocol is not shown in this experiment due to its higher values of average-end-to-end delay.

5.2 Shortest Journey and the Number of Hops

The goal of the EG shortest journey algorithm is to calculate journeys that reach the destination using the minimum number of hops during certain period of time. This behavior is shown in Fig. 5, on which the EGShortest algorithm has the minimum values of average hop-count. If we consider the intersection of packets reached by all simulated protocols, the EGShortest could be **The IFQ queue is at the link layer, i.e., it is used when the routing layer wants to effectively send a packet to be delivered.**
used as lower bound in this metric. Following the low values of hop-count shown above, another important characteristic of the \textit{EG}_{foremost} is the low values of energy consumption, which also showed to be a reference curve. /*JM: ALTEREI ESSA ULTIMA FRASE*/

In the case of reactive protocols, the cache time parameter plays an important role here. The high values of hop-count on these protocols are related to cached routes /*JM: NOVO PARAGRAFO, TENHO QUE REVISÁ-LO... TALVEZ*/
that are not updated to be the shortest. The proactive protocols, otherwise expected, reached values close to the $E_G_{Shortest}$, which means that the interval of its topology updates are accurate enough. However, it is important to note again that we are taking into account only the packets that are delivered by all protocols.

5.3 Bottlenecks and Congestion

Despite the expected good values obtained by the EG protocols in their respective metric, end-to-end delay and number of hops, that show how good other protocols could perform, the EG protocols drop rate are almost zero for all the evaluated values of HoldTime and PauseTime. In Fig. 6, the values of $E_G_{Foremost}$ and $E_G_{Shortest}$ in the Intermittent Model scenarios are very close to each other, actually one curve is on top of another, and their values are close to zero. This metric means that a route exists between the source and desatination almost all the time.

![Fig. 6. Number of dropped packets by no available route in the Intermittent Model scenarios.](image)

However, during the experiments, we notice that at some scenarios the number of dropped packets by the $E_G_{Foremost}$ and $E_G_{Shortest}$ were higher than expected. The intrinsic behavior of EG protocols, which schedule packets to be sent when some connections are established, yields to the problem of bottlenecks. Since a large quantity of packets are scheduled to be sent at the same moment, the link interface queue (IFQ) cannot hold that incoming traffic (its default size is 50 packets and in some experiments is raised to 500 packets). /*JM:ADICIONEI — NOVAMENTE UMA MENCAO AO TAMANHO 500*/

Furthermore, the EG algorithm do not have any mechanism to balance the flows and many flows with different sources could potentially use the same path, even when some other ones are available at the same cost (i.e. arriving at the
same time). This characteristic appeared in the low connectivity and low dynamics scenarios of the Intermittent Model, in which the nodes in the evolving graph remain disconnected for long time periods, and a large quantity of packets are then scheduled to the moment when these nodes wake-up.

The histogram in Fig. 7 shows the number of dropped packets over time for one single simulation, namely with SleepProb at 50% and HoldTime at 180s in the Intermittent Model. It shows that in EG protocols the packets are dropped in a burst, again because important nodes go to sleep for a long time and when they wake-up, a large quantity of packets are waiting to be sent, overflowing the queues that then drop the packets. /*JM: DROP ME PARECEU OK AQUI*/

![Fig. 7. Number of dropped packets over time. The peaks are the EG based protocols.](image)

In Fig. 8 we see the high values of dropped packets by IFQ overflow on a low connectivity scenario (28% of packets are dropped by IFQ at SleepProb 50%). These high values negatively affect the throughput of EG protocols, as mentioned above.

### 5.4 Reducing congestion in EG algorithms

The discussion above shows the importance of managing the flows of data during time, even when using EG protocols as a reference. Unfortunately, balancing flows in evolving graphs is still an open problem in Graph Theory. Therefore, we tried three different empirical approaches to address the packet dropping problem caused by bottlenecks in EG algorithms (see Table 1, below):

1. **Enforced Jitter**: Add an enforced jitter (a random value uniformly chosen between 0 to 0.5 seconds) at the sending time to each packet;
2. **SmartJitter**: Add a fixed size jitter only when some connection is overflown. The size of this jitter is the same as the average edge traversal time;

3. **Increase the IFQ length**: Raise buffer size of the interface queue from 50 to 500 packets.

<table>
<thead>
<tr>
<th>Packet</th>
<th>IFQlen</th>
<th>Jitter 0.5s</th>
<th>SmartJitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t$</td>
<td>$t$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t$</td>
<td>$t$</td>
</tr>
<tr>
<td>...</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t$</td>
<td>$t$</td>
</tr>
<tr>
<td>$p_{50}$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t$</td>
<td>$t$</td>
</tr>
<tr>
<td>$p_{51}$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t + \delta$</td>
<td>$t$</td>
</tr>
<tr>
<td>$p_{52}$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t + 2 \times \delta$</td>
<td>$t$</td>
</tr>
<tr>
<td>$p_n$</td>
<td>$t$</td>
<td>$t + \text{rnd}(0.5) t + (n - \text{IFQlen}) \times \delta$</td>
<td>$t$</td>
</tr>
</tbody>
</table>

The results of the experiments with these three approaches can be seen in Fig. 8. IfqLen 50 is the reference curve. The first approach is the enforced random chosen jitter when sending each packet at each node, ranging from 0 to half of a second (0.5s). In the average, the number of dropped packets decreased 37%. One drawback of this approach is the high values of end-to-end delay, due to the extra time added at each scheduled packet.

**Fig. 8.** Different approaches to minimize the drop by IFQ (enforced jitter, SmartJitter and IfqLen).
The second approach, the SmartJitter, is an improvement of the former. Here, when the queues (one for each pair of neighbours) are not full, the nodes can send packets to other nodes without any jitter on sending time. However, when some node fills the IFQ (number of waiting packets greater than 50), then a fixed size jitter is added to each subsequent packet to be sent, in a cumulative way. The calculation of the SmartJitter time at node $u$ sending a packet to $v$ at time $t_v$ is shown in the following schema:

\[
\text{if } npkts[t_v] \leq IFQlen \text{ then} \\
\quad \text{smartjitter} = 0 \\
\text{else} \\
\quad \text{smartjitter} = \delta \ast (npkts[t_v] - IFQlen) \\
\text{end if} \\
npkts[t_v] = npkts[t_v] + 1
\]

At the end, the sent time will be $t_v = t_v + \text{smartjitter}$ and $npkts[t_v]$ is the number of packets already scheduled to use the edge from $u$ to $v$ at time $t_v$. $IFQlen$ is the interface queue length, and $\delta$ is the average traversal time of one-hop transmission. We used the value 3.6 ms in all simulations, which was estimated from the average value of one-hop transmission with packet size of 256 bytes. The value of the traversal time is linearly dependent on the size of the packet. This value was obtained through simulations using similar network loads. The traversal time used in the EG foremost algorithm is the same estimated $\delta$ value.

With the SmartJitter, the number of dropped packets decreased 52% compared to the original $EG_{Foremost}$. Furthermore, we note that the best solution is to increase the default length of the IFQ from 50 to 500 packets. In this case, the values of dropped packets decrease 89%. However, we showed that the SmartJitter is a good solution when the size of IFQ buffer cannot be changed.

6 Conclusions and Future Work

In this work we managed to implement in practice some theoretical algorithms, reinforcing their interest and raising some practical problems. We were able to propose experimental ways to effectively solve some of them, even without the support of the theory, as it remains an open problem. Our main contribution in this work was to show that an $EG$ based routing protocol is well suited for networks with known connectivity patterns, and that the model as a whole may be a powerful tool for the development of routing protocols, even in practical scenarios.

This first implementation of an $EG$ based protocols opens some avenues for further detailed research. For instance, a routing bottleneck appears when most used nodes become unavailable for a long period of time, causing overhead when they reappear in the network, leading to packets being dropped due to collision and queue overflows. The use of high values of enforced jitter time when sending
packets can minimize the drop rate, but is not feasible in regular protocols. We introduced the SmartJitter as an option to minimize the congestion and achieved good results for a problem where there is still no theoretical results (flows over EG). The development of a good EG adaptive algorithm could possibly manage this problem, anticipating congested nodes in order to find out alternative paths.

Future work includes implementation of other EG based protocols with different metrics, like fastest delay. A natural extension to this work is related to the deviations in the predicted network dynamics, on which the actual EG used by the nodes is not accurate anymore. This engenders the utilization of a model with stochastically predictable behaviour to better address such variations.

Acknowledgments

The author would like to thank Afonso Ferreira for his help in the development of this work.

References


