Improving Rumor Spreading

Quentin Dufour
advised by David Bromberg and Davide Frey

Univ Rennes, Inria, CNRS, IRISA

October 1, 2018
1 Introduction to Rumor Spreading

2 Multisource Rumor Spreading with Network Coding

3 Rumor Spreading Adaptiveness: Adjust Pull Frequency
What achieves Rumor Spreading?

Deliver a message to every node of a group with high probability.

Nodes in blue know the message
Use cases for Rumor Spreading

- Collaborative Platforms
  - Chat, Communication Platform (eg. Rumor Monger)
  - Collaborative Editors (eg. CRATE)
- Collaborative News Feeds (could replace WebSub or Atom/RSS)
- Sensor Network
- Membership

Basically any publish/subscribe mechanism
Naive solution

At the beginning, only D knows the message. 
**D sends the message to every nodes of the network.**
Naive solution

At the beginning, only D knows the message. **D sends the message to every nodes of the network.**
Make nodes collaborate: a push only protocol

At the beginning, only D knows the message. Each nodes forward the message 2 times to a random node.
Make nodes collaborate: a push only protocol

At the beginning, only D knows the message. Each nodes forward the message 2 times to a random node.

Infect and Die with fanout = 2
Make nodes collaborate: a push only protocol

At the beginning, only D knows the message. Each nodes forward the message 2 times to a random node.

Infect and Die with fanout $= 2$
Make nodes collaborate: a push only protocol

At the beginning, only D knows the message.
Each nodes forward the message 2 times to a random node.

Infect and Die with fanout = 2
Ask for the message: a push-pull protocol

It is costly to reach every node with high probability. Adding a pull phase can help last nodes retrieve the message.
Ask for the message: a push-pull protocol

It is costly to reach every node with high probability. Adding a pull phase can help last nodes retrieve the message.
Ask for the message: a push-pull protocol

It is costly to reach every nodes with high probability. Adding a pull phase can help last nodes retrieve the message.

Presented protocols handle only one message.
Working with multiple messages

With $P$ being the payload of a message, how to know if it’s a new message? How to discover missing messages?

A → send $P$ → B → send $P$ → C
Working with multiple messages

With \( P \) being the payload of a message, how to know if it’s a new message? How to discover missing messages?

- Add a unique identifier \( I_i \) to every messages

\[ \text{send } I_3, P \]

\[ \text{send } I_4, P \]
Working with multiple messages

With $P$ being the payload of a message, how to know if it’s a new message? How to discover missing messages?

- Add a unique identifier $I_i$ to every message
- Piggyback a list of known identifiers on every “send” messages

A $\xrightarrow{\text{send } [I_1, I_2, I_3, P]}$ B $\xleftarrow{\text{send } [I_1, I_4, I_3, P]}$ C
Working with multiple messages

With $P$ being the payload of a message, how to know if it’s a new message? How to discover missing messages?

- Add a unique identifier $I_i$ to every messages
- Piggyback a list of known identifiers on every ”send” messages
- Ask a message list during pull

ask $[I_1, I_2, I_4]$
Working with multiple messages

With $P$ being the payload of a message, how to know if it’s a new message? How to discover missing messages?

- Add a unique identifier $I_i$ to every messages
- Piggyback a list of known identifiers on every ”send” messages
- Ask a message list during pull

```
ask [I_1, I_2, I_4]  
send [I_3, I_4], I_1, P
```
Encode messages to improve throughput and latency

B already knows one message, either $l_1, P_1$ or $l_2, P_2$.

A \quad \text{send $P_1$? send $P_2$?} \quad B
Encode messages to improve throughput and latency

B already knows one message, either $I_1, P_1$ or $I_2, P_2$.

Encoding messages increase the chance of providing new information.
Encode messages to improve throughput and latency

B already knows one message, either $I_1, P_1$ or $I_2, P_2$.

Encoding messages increase the chance of providing new information.

It can be expressed over a Galois Field.
Encode messages to improve throughput and latency

B already knows one message, either $l_1, P_1$ or $l_2, P_2$.

A

send $\emptyset, \{l_1 : 1, l_2 : 1\}, X_{P_1, P_2}$

B

- Encoding messages increase the chance of providing new information.
- It can be expressed over a Galois Field
- Identifier is replaced by a list of identifiers and their associated coefficient. $X_{P_1, P_2} = P_1 \oplus P_2$ in this case.
Generalize with Random Linear Network Coding

Random coefficients with independent linear combination?
Use a bigger Galois Field. Example with GF(256):

A

B

D

C

E

Choose random coefficients:

\{I_1: 16, I_2: 3, I_3: 21\}

Compute new payload:

\[X = 16P_1 + 3P_2 + 21P_3\] over GF(256)

Send the encoded message
Generalize with Random Linear Network Coding

Random coefficients with independent linear combination?
Use a bigger Galois Field. Example with GF(256):

Choose random coefficients:

\[ \{I_1 : 1\}, P_1 \]
\[ \{I_2 : 1\}, P_2 \]
\[ \{I_3 : 1\}, P_3 \]

Compute new payload:

\[ X = 16P_1 + 3P_2 + 21P_3 \] over GF(256)

Send the encoded message
Generalize with Random Linear Network Coding

Random coefficients with independant linear combination?
Use a bigger Galois Field. Example with GF(256):

Choose random coefficients: \( \{l_1 : 16, l_2 : 3, l_3 : 21\} \)
Generalize with Random Linear Network Coding

Random coefficients with independent linear combination?
Use a bigger Galois Field. Example with GF(256):

Choose random coefficients: \( \{ I_1 : 16, I_2 : 3, I_3 : 21 \} \)
Compute new payload: \( X = 16P_1 + 3P_2 + 21P_3 \) over GF(256)
Generalize with Random Linear Network Coding

Random coefficients with independent linear combination?
Use a bigger Galois Field. Example with GF(256):

Choose random coefficients: \( \{ l_1 : 16, l_2 : 3, l_3 : 21 \} \)
Compute new payload: \( X = 16P_1 + 3P_2 + 21P_3 \) over GF(256)
Send the encoded message
Group messages in generations

Coding explosion:

\{ l_1 : 16, l_2 : 3, ... , l_{1501} : 15 \}, \emptyset, X
Group messages in generations

Encoding is only done between message of the same generation:

\[
G_1, \{ I_1 : 16, I_2 : 3 \}, \emptyset, X
\]

\[
G_{56}, \{ I_{1500} : 87, I_{1501} : 15 \}, \emptyset, X'
\]

How to assign a generation to a message?

- Encode messages that are emitted in the same timeframe
- Every node starts with a logical clock at zero
- When a node emits a message, it will assign its clock as message generation then increase its clock
- If a node forwards a message tagged with a generation bigger than its local clock, it updates its clock to match the generation it has seen
Comparison against Pulp, a push-pull algorithm

Figure: Pulp original

Figure: Our algorithm
Why pull frequency must be adapted

Figure: Too fast

Figure: Too slow
Can we use control theory?

We want:

- to control
  - pull usefulness (number of useful pull / number of total pull)
  - delivery delay (by monitoring known missing messages evolution)
- to take action on pull frequency

Looked at:

- Open Loop vs Closed Loop
- Binary Feedback (used in TCP Congestion Control)
  - Additive (or Multiplicative) Increase, Additive (or Multiplicative) Decrease
- Proportional-Integral-Derivative (PID) Controller

Problem: unstable environment
(message rate, loss, churn, network size, latency)

- Adaptive Controller
- System Identification
Pulp adaptiveness analysis

Reminder: control usefulness and missing messages, act on pull frequency

Take a sample every $\Delta \text{adjust}$ time

2 strategies:

- $\Delta \text{missing}$ increased
  - Apply this formula (open loop):
    \[
    \Delta \text{pull} = \frac{\Delta \text{adjust}}{\Delta \text{missing} + \text{useful}}
    \]

- otherwise, apply Multiplicative Increase Multiplicative Decrease
  - if usefulness $> 0.5$, decrease $\Delta \text{pull}$
  - otherwise increase $\Delta \text{pull}$
First experimentations

Start at 1.5 msg/s, then emit 150 msg/s, then go back to 1.5 msg/s
We graph usefulness (useful pull / total pull) over time

(a) Pulp

(b) MIMD

(c) PID
Conclusion and future works

Goal: provide a versatile Gossip-based dissemination brick by:
- Improving performances
- Improving ease of use

Future work:
- Better handle unstable environment
- Provide adaptiveness on more parameters
- Create a real world implementation
- Model problem, provide bounds, etc.