Decentralized Systems Engineering

CS-438 – Fall 2024

DEDIS

Pierluca Borsò-Tan and Bryan Ford

EPFL

Credits: P. Tennage, C. Basescu, et al.

Decentralized Communication

Gossip

(Homework 1)

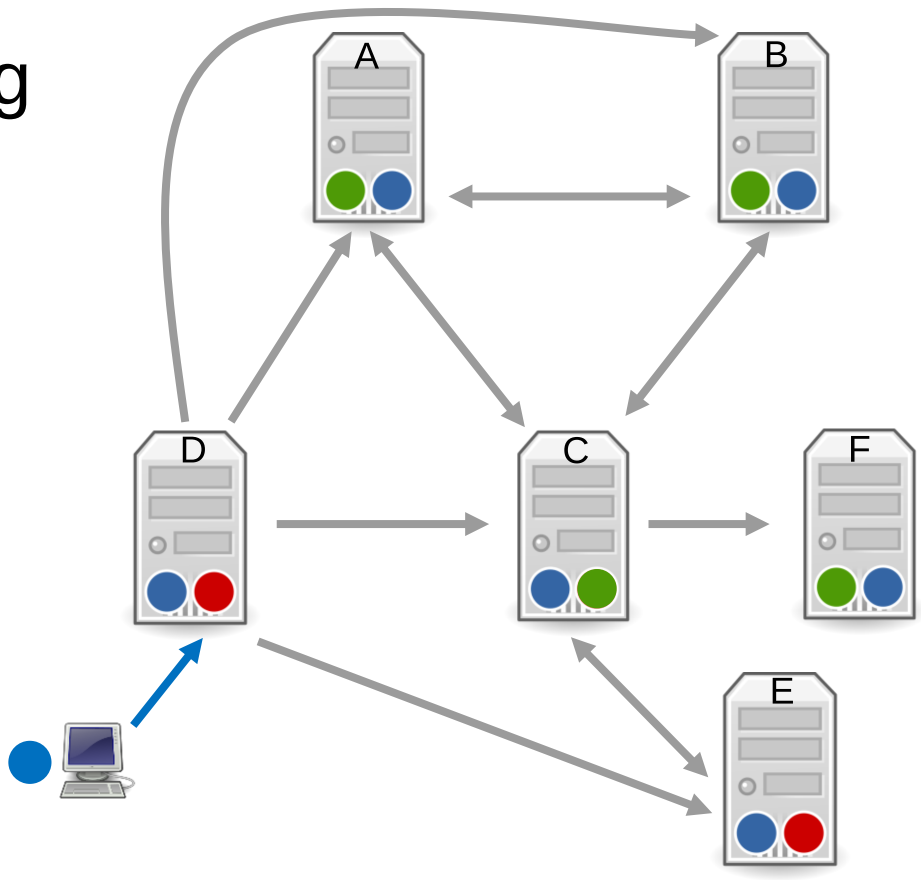
Recap: Improved Gossiping

What can we learn from people ?
Rumor mongering

```
On receiving message M:

  pick random neighbor, send M
  neighbor replies: new rumor?

  if new:    repeat
  else:      flip_coin()
             if head: repeat
             else:    stop
```



What's good about this ? Which issues do you foresee ?

Recap: Improved Gossiping (cont'd)

How can make sure messages reach *every* node?

Anti-entropy

```
Periodically (when timer fires):
  pick random neighbor
  send "anything new?"
  reduce entropy (ihave/sendme)
```

What's good about this ? What's limiting ?

This (slowly) ensures <u>complete</u> dissemination, at $O(n)$ per period

Case study: Twitter

Gossip relies on message-IDs.

How do you pick a good Message-ID in a (centralized) distributed setting ?

- For most applications, they need to be sortable
- Globally unique, even in distributed settings

→ Snowflake IDs

[0 (1 bit) | timestamp (41 bits) | machine ID (10 bits) | counter (12 bits)]

Announced 2010, adopted by Instagram (2012), Discord (2015)

Case study: Mastodon

Snowflake IDs (64 bits)

[0 (1 bit) | timestamp (41 bits) | machine ID (10 bits) | counter (12 bits)]

What about a *decentralized* setting?

- Would this work?
- What else do we need to do ?

Case study: Mastodon

```
{

'created_at': datetime.datetime(2022, 11, 13, 0, 52, 37, tzinfo=tzutc()),

'id': 109347716491680514,

... message data ...

'uri': 'https://example.com/@user/109347716173491502'

}
```

Local Snowflake ID

Original URI w/ Snowflake ID

Gossip Quality Measures

Is your gossip any good ?
How can you tell ?

- Residue
- Traffic $\frac{total\ traffic}{number\ of\ nodes}$
- t_{avg} , t_{95} average & 95th percentile time for rumor → node
- t_{last} time for rumor → last node

Broadcast	Rumor-Mongering	Anti-Entropy
	Early: low Late: high	Early: high Late: low

What are the parameters & trade-offs ?

Rumor-mongering (fast, randomized):

- Feedback vs. blind
- Randomized vs. counter
- Constant: probability / counter

Anti-entropy (slow, complete):

- Periodicity

How can we “delete” rumors ?

Death Certificates

- When is the rumor deleted ?
- How does this affect propagation ?
- Why can rumors resurrect ?
- When can we delete the death certificate ?
- What if a server is offline ?
    - “Dormant” death certificates
    - Phased deletion

A few applications of Gossip

- Metadata propagation
- Failure detection
- Group membership

Example

- Apache Cassandra
- CockroachDB
- Consul

Communicating with (many, unknown) peers

You now understand:

- How do we reach unknown peers ?
- How do we eventually reach *every* peer ?
- How can we communicate reliably ?
  - Are they online ?
  - Is the network « stable » ?
- How do they find out about us / our node ?

Decentralized Search

Finding Data

(Homework 2)

Finding data among (many, unknown) peers

- Same machine
  build a local index, and/or Linux `locate`, `find`, `grep` commands

- Local networking
- Global networking, centralized trust
  crawling, processing, indexing, then using the (distributed) index…
  Google ☺

- Decentralized
  ??? → today’s and next week’s lecture

Finding data among (many, unknown) peers

Many open questions:

- Where can the data be found ?
- Does the data even exist ?
- Who knows about it ?
- How do we retrieve it ?

Most importantly:

- What can we assume about the peers and the network ?

Distributed search algorithms

Two big families:

- Unstructured search
    - Robust to churn, instantly adaptive

    → Today's lecture

- Structured search
    - Much more efficient, many more problems

    → Next week's lecture

Building Gnutella

Context (1999 – 2008):

- No Spotify, no Netflix
- No BitTorrent
- People still want entertainment

Specifications:

- Search any file, anywhere
- Metadata can be searched
- Complex queries are allowed ((A or B) and C)
  e.g. artist is “Bob Marley” and title contains “birds”

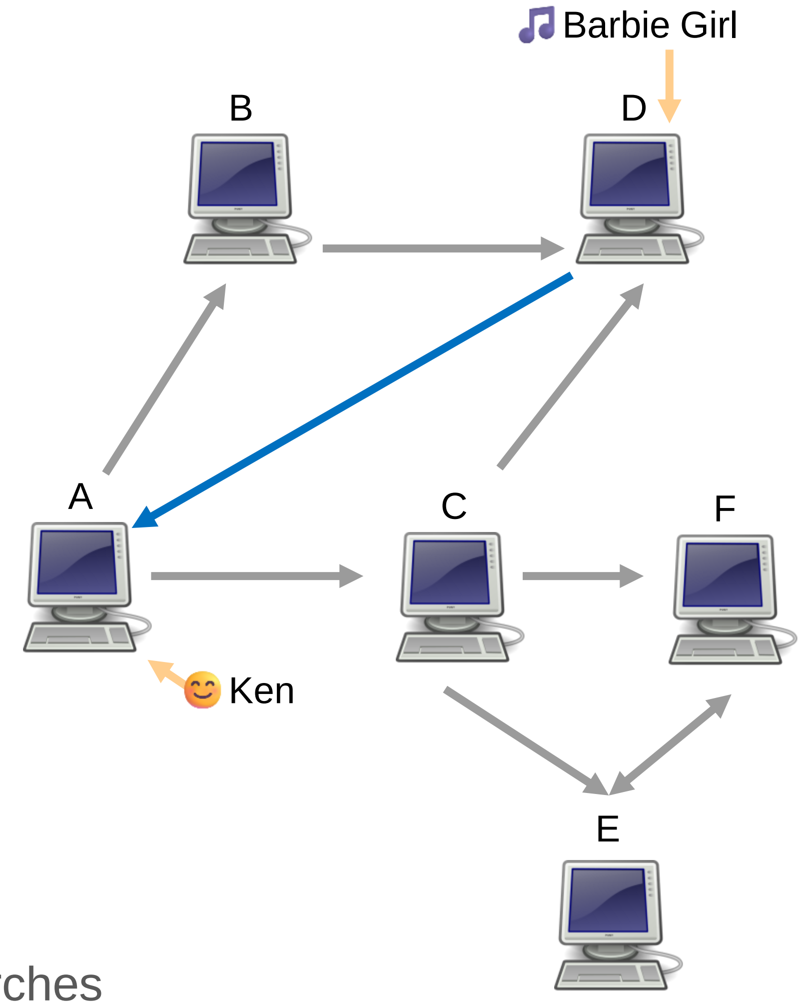
Standard, basic algorithm

What can we learn from people ?
Flooding

- Gossip searches (query)
- Direct response (query hit)

Which issues do you foresee ?

- Unpredictable delays
- Connectivity
- Efficiency : all nodes see & process all searches



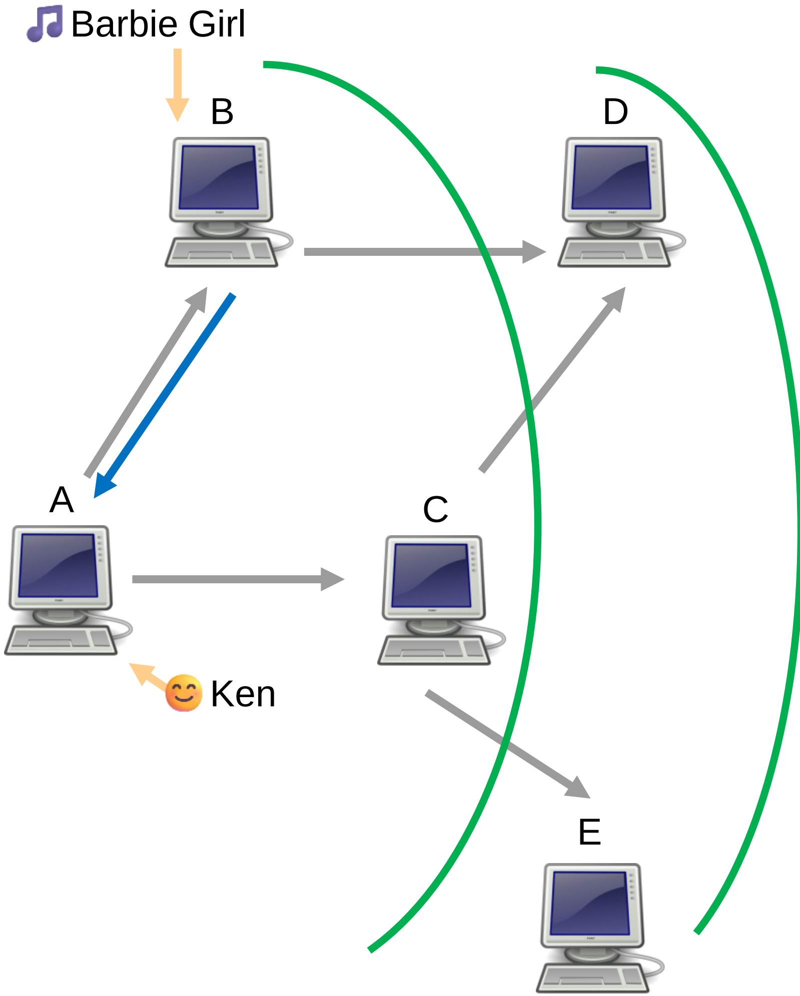
Optimizations

Can we *not* flood everyone ?
Expanding-ring search

- Limited flooding (TTL)
- Increasing TTL on retry

What are the trade-offs ?
- Higher latency
- Asymptotically worse $O(n \log n)$

- Pragmatically, it works !



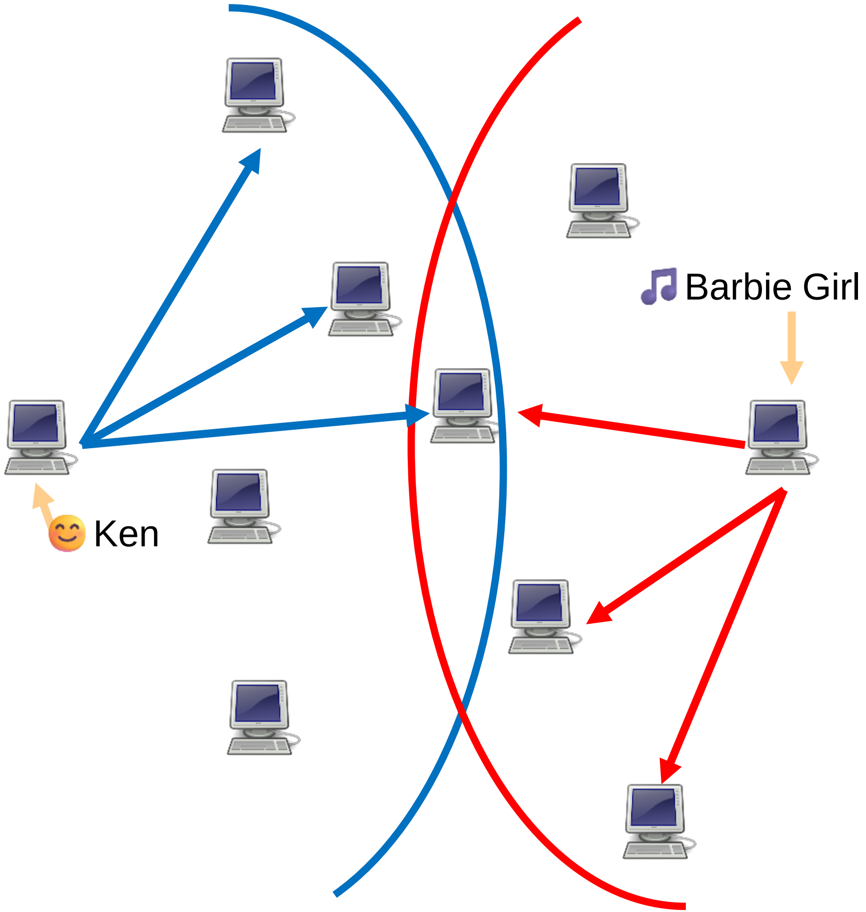
Optimizations

Can we make it more efficient ?
BubbleStorm

- Birthday paradox
- Data search & storage
  random “meet in the middle”

Key considerations:

- Asymptotically efficient $O(\sqrt{n})$
- “Mostly unstructured”
- Tunable parameters
- Extremely robust / resilient



Next steps

Reading on Moodle:

- (Mandatory) BubbleStorm: Resilient, Probabilistic, & Exhaustive P2P Search

→ Use Friday's session to ask questions