LECTURE 19: SCALING MICROBLOGGING SERVICES WITH DIVERGENT TRAFFIC DEMANDS

COMP4641: Social Information Network Analysis and Engineering

Friday May 8th 2015

Outline

- Background and Related Work
- Measuring Availability at High Load
- Key Design Rationales of Cuckoo
- Performance Evaluation
- Conclusion

Background

Microblogging services have become a significant new form of Internet communication utility!

❑ Take Twitter as an example:

1. **Less than 10 years (launched in October 2006)**
2. **300+ million monthly active users;**
   - userbase is still growing
3. **500+ million tweets posted per day**

Twitter **Google Buzz**

Plurk **Sina**

Usage of Microblogging (1)

- Originally designed as an online social network (OSN) with quick updates

- Evolved to encompass a wide variety of applications, e.g.,

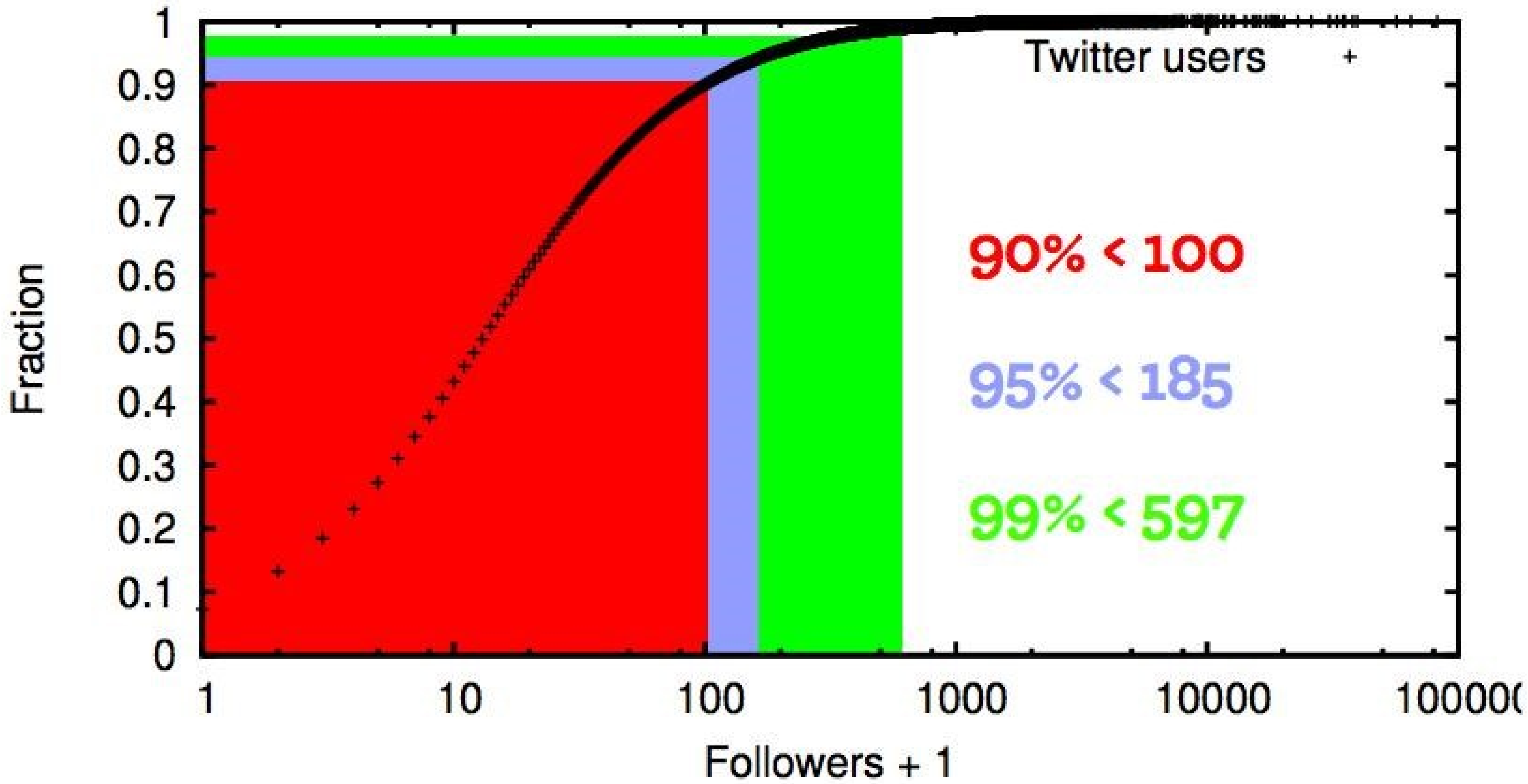
1. Predict gross revenue for movie openings
2. Produce predictions for election results
3. Sensors for Internet service failure
4. Real-time warning system for earthquakes

Usage of Microblogging (2)

Microblogging usage has also evolved significantly over time!

- **Besides as a social communication tool, much of traffic today is communication from celebrities and news media to their fans and followers.**

CDF OF TWITTER FOLLOWERS*



*D. R. Sandler et al., Bird of a FETHR: Open, decentralized micropublishing, IPTPS 2009.

There are a few highly-subscribed celebrities.

Katy Perry has 69.3 million followers

- **Twitter serves more as an information spreading medium than an online social network service*.**

*H. Kwak et al., What is Twitter, a Social Network or a News Media? WWW 2010.

Twitter Top Followers

	Twitter users		Followers	Following	Tweets
1	KATY PERRY	@katyperry	69,391,854	156	6,481
2	Justin Bieber	@justinbieber	63,354,636	208,105	28,516
3	Barack Obama	@BarackObama	58,905,953	643,092	13,456
4	Taylor Swift	@taylorswift13	57,198,309	197	3,350
5	YouTube	@YouTube	50,643,096	889	13,607
6	Lady Gaga	@ladygaga	46,316,046	132,686	6,522
7	Justin Timberlake	@jtimberlake	45,052,133	106	2,938
8	Rihanna	@rihanna	44,979,193	1,176	9,688
9	Ellen DeGeneres	@TheEllenShow	43,034,454	39,008	10,522
10	Britney Spears	@britneyspears	41,762,034	398,970	4,112
11	Instagram	@instagram	38,699,862	3	6,206
12	Twitter	@twitter	38,222,422	103	1,984

Usage of Microblogging (3)

- These major sources of traffic have a very tangible impact on the performance and availability of microblogging as a service.

1. Loss in availability
   - Malicious attacks
   - Hardware failures
2. Traffic overload and flash crowds

Twitter's Short-Term Solutions

1. **Per-user request and connection rate limits**
   - Rate limit
     - Only allows clients to make a limited number of calls in a given hour
     - Twitter: 150 requests per hour, 2,000 requests for whitelist
   - Upper limit on the number of people a user could follow
     - Orkut: 1000, Flickr: 3000, Facebook: 5000,
     - Twitter: 2000 before 2009

2. **Network usage monitoring**

3. **Doubling the capacity of internal networks**

It is clearly challenging and likely costly to scale up with demand in using the current centralized architecture.

Decentralized Microblogging System

FETHR (IPTPS 2009)

- **Full decentralization**
  - Users directly contact each other via HTTP
- **Propose to use gossip for popular content propagation**

➢ **Cannot guarantee data delivery**
  - Some tweets cannot get to users due to user access asynchronism

➢ **Does not elaborate gossip component nor implementation in prototype**

➢ **No heterogeneous client support**

➢ **No user lookup service**

Outline

- Background and Related Work
- **Measuring Availability at High Load**
- Key Design Rationales of Cuckoo
- Performance Evaluation
- Conclusion

Measurement Study

1. 20-day Twitter availability and performability measurement
   - Including the period of FIFA World Cup 2010

2. User behavior analysis for over 3,000,000 Twitter users

3. Measurement of system scalability of a generic centralized microblogging architecture

Availability and Performability of Twitter (1)

- **Measurement period: Jun. 4 – Jul. 18 2010**
  - Including FIFA World Cup 2010
- **Measurement place: a city in Germany**
  - The same time zone (CEST) as South Africa
- **Availability in terms of service rejection rate**
  - Randomly select a Twitter user and request for his recent 200 tweets in every 5 seconds
- **Performability* in terms of response latency**
  - Upload latency: sending tweets to the Twitter server
  - Download latency: request the uploaded tweets from Twitter

* P. M. Broadwell, Response Time as a Performability Metric for Online Services. Tech. Rep. UCB/CSD-04-1324, University of California at Berkeley, 2004.

Availability and Performability of Twitter (2)

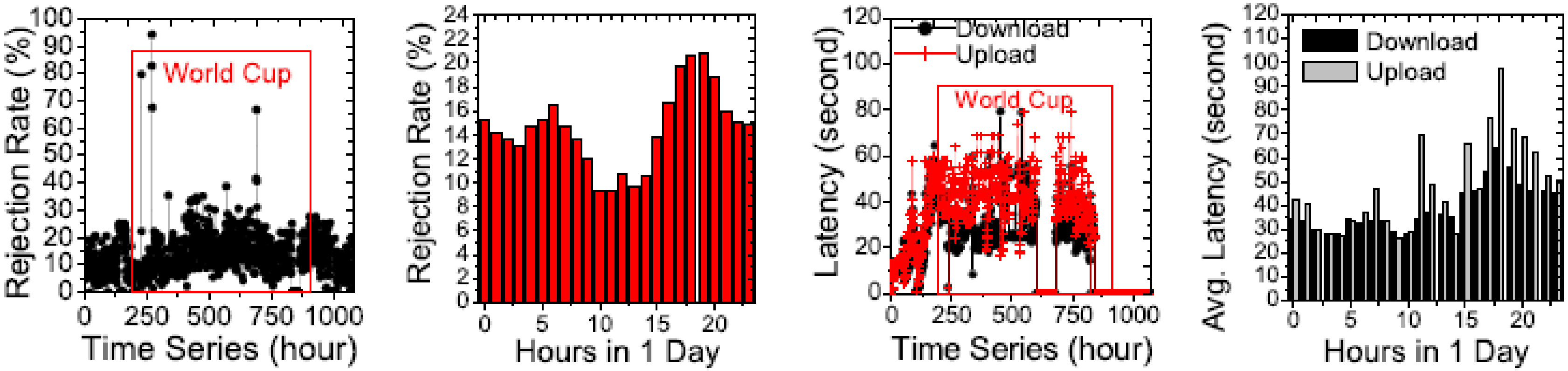


Fig. 1. Twitter measurement: (a), (c) Service rejection rate, response latency in time series (Jun. 4 to Jul. 18, 2010); (b), (d) Service rejection rate, response latency for 24 hours.

❑ Results

1. **Twitter's availability is poor (even at normal time)**
2. **The flash crowd has an obvious impact on availability (service rejection rate) and performability (response latency)**

User Access Pattern Analysis (1)

- **Analysis of large-scale Twitter user trace**
  - 3,117,750 users' profile, social links, tweets
  - 4 machines with whitelisted IPs
  - Snowball crawling began with most popular 20 users reported in [1] using Twitter API
- **Consider two built-in Twitter's interaction models**
  - POST and REQUEST

[1] H. Kwak et al., What is Twitter, a Social Network or a News Media? WWW 2010.

User Access Pattern Analysis (2)

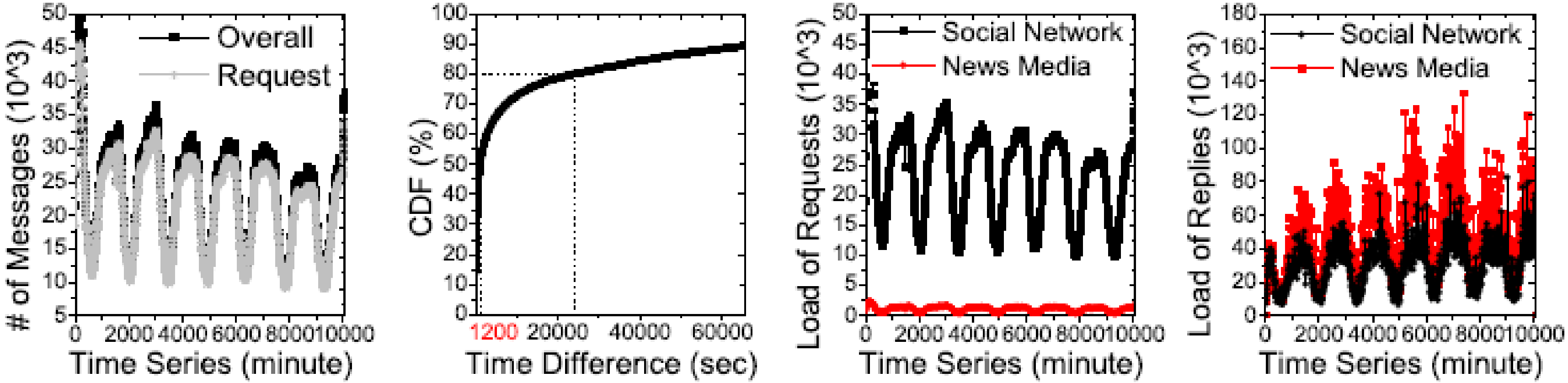


Fig. 2. User access patterns: (a) # of request messages to the servers; (b) Time differences of two adjacent tweets; (c), (d) Server load in terms of received, replied messages.

❑ Results

1. **Request messages make up the dominating traffic proportion**
2. **There are still 50% of time differences larger than 1200 second and 20% larger than 24,000 second.**
3. **Although news media usage holds small proportion of incoming server load, it occupied a great proportion of outgoing server load.**

Scalability of a Generic Centralized Microblogging System (1)

- Treat Twitter as a black box and reverse engineer its operation based on Twitter traces
  - Still consider POST and REQUEST as the main interaction models
- Each user interaction is implemented through one or more connections with centralized servers
- Use BFS to build four datasets for 10,000, 30,000, 50,000, 100,000 user scale and compare the server performance under different user scales

Scalability of a Generic Centralized Microblogging System (2)

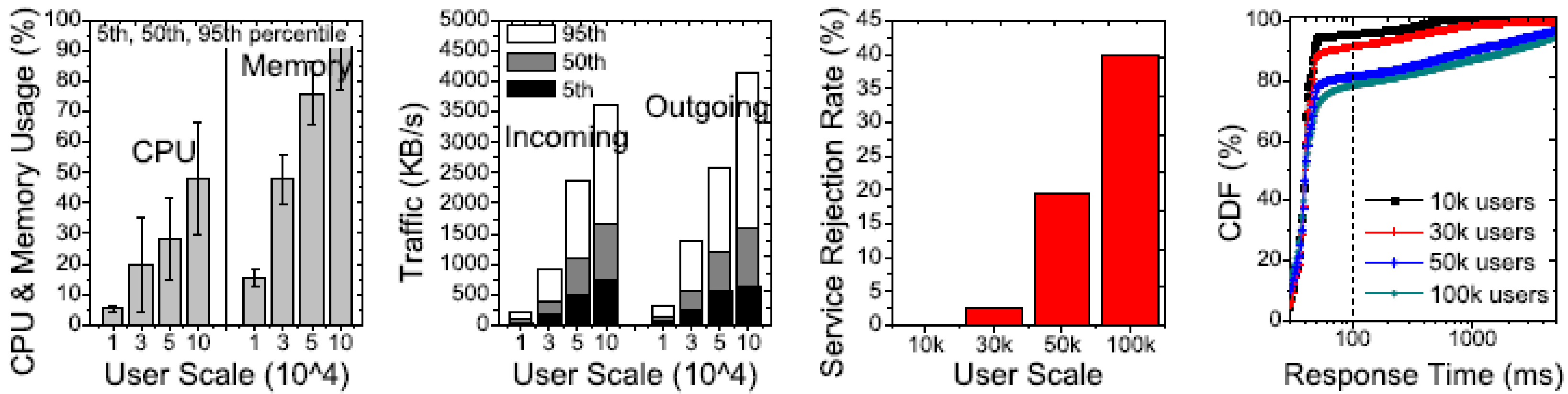


Fig. 3. Scalability of the centralized microblogging architecture: (a) CPU and memory usage; (b) Traffic usage; (c) Service rejection rate; (d) Response latency.

❑ Results

1. **Linear growth of CPU/memory usage and traffic usage**
2. **High server rejection rate when servers are overloaded**
3. **Long response latency when servers are overloaded**

Three Key Observations

1. The centralized architecture has limited scalability with the increasing number of users.

2. The main server load and traffic waste are caused by polling requests.

3. The social network and news media components of microblogging have distinct traffic patterns and their mix makes the system hard to scale.

Outline

- Background and Related Work
- Measuring Availability at High Load
- Key Design Rationales of Cuckoo
- Performance Evaluation
- Conclusion

Decoupling the Two Components (1)

- **The biggest reason that microblogging systems like Twitter do not scale:**

- **They are being used as both a social network and a news media infrastructure at the same time!**

  1. **The two components have very different traffic and workload patterns with different dissemination models**
     - **Social network**: great incoming traffic, a few followers, not very active in updating statuses
     - **News media**: great outgoing traffic, huge numbers of followers, highly active in posting news
  2. **There is no single dissemination mechanism can really address both two at the same time.**

Decoupling the Two Components (2)

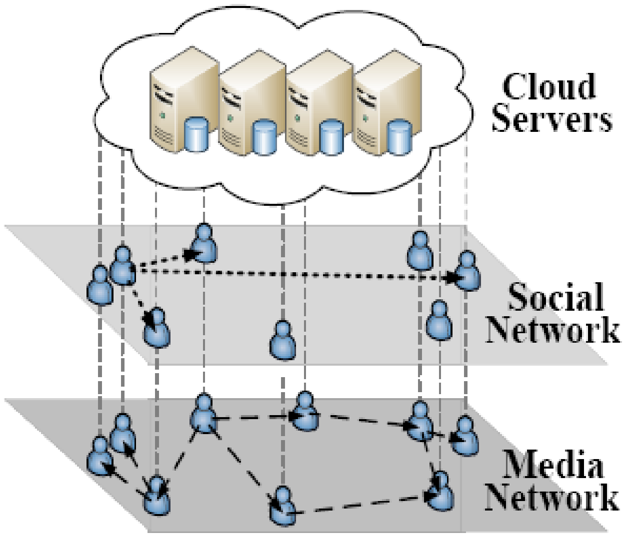


Fig. 4. Cuckoo architecture

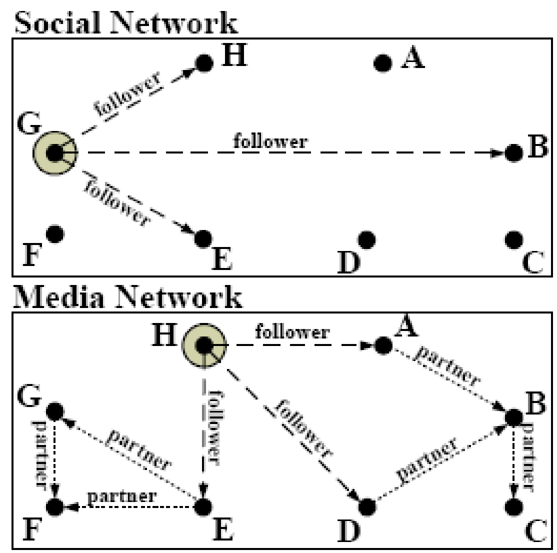


Fig. 5. Complementary content delivery

- **Complementary delivery mechanisms**
  - **For social network, directly push contents to followers via unicast (form a social network)**
  - **For news media, use gossip to provide highly reliable and load-balanced content dissemination (media network)**

Combination of Server Cloud and Client Peers (1)

- The centralized architecture is hard to scale!
  - High server load
  - Performance bottleneck
  - Central point of failure

- Truly decentralized P2P systems have earned notoriety for
  - Difficult for coping with availability and consistency
  - No guarantee on data delivery

Combination of Server Cloud and Client Peers (2)

Incorporate the advantage of both centralized and decentralized architectures:

- **Server cloud (a small server base)**
  - Ensure high data availability
  - Maintain asynchronous consistency
  - Host all the user contents

- **Cuckoo peers (client peers at network edge)**
  - Data delivery/dissemination
    - Abandon polling -> abandon main server burden
  - Decentralized user lookup

Social Relations

Each user maintains three kinds of social relations:

- **Followee**
  - User profiles

Form the social network

- **Follower**
  - User profiles
  - Node Handler (NH)
    - The necessary information for connection (e.g., IP, port)
  - Maintain a logarithmic subset if # of followers is huge

- **Partner**
  - Form the media network
  - For each followee, a set of partner information
    - DHT-based random walk for updating

Follow: Build Social Relations

The "Follow" operation explicitly build the followee-follower relations between user pairs.

- **First lookup the followee's NH and try to contact**
  - ***Followee online***
    1. **<u>Followee & Follower:</u> build social links mutually**
    2. **<u>Follower:</u> inform the server cloud of the built relation**
  - ***Followee offline***
    1. **<u>Follower:</u> submit the willing to server cloud**
    2. **<u>Server cloud:</u> check the validity and reply**
    3. **<u>Followee:</u> check the inconsistency and contact the follower as compensation**

Unicast Delivery for the Social Network

- **Post a tweet => direct push to each follower**
  - Serial unicast socket
  - Via NH
- **Inform followees when changing NH**
  - Direct inform
  - Upload to the server cloud
- **Regain missing tweets in offline period**
  - From the server cloud
  - Efficient inconsistency checking
    - Based on the well-structured timeline (reverse chronology)

Gossip Dissemination for the Media Network

- **Enable interested users involved in the micronews dissemination process**
  - **Gossip-based information dissemination**
    - Scalable, resilient to network dynamics, load-balanced
    - Maintain # of partners (fanout) to be logarithmic of # of followers
- **DHT-based partner collection**
  - **Announcement**
    - "Say Hello" to random destinations
    - Overhearing the hello message passing by
  - **Discovery**
    - DHT-based random walk
- **"Infect-and-Die" model**
  - **Once infected, remain infectious for one round precisely before dying**

Support for Client Heterogeneity (1)

Differentiate user clients into three categories:

- **Cuckoo-Comp**
  - Stable nodes
  - Construct DHT and provide DHT-based user lookup
  - Participate in message dissemination
- **Cuckoo-Lite**
  - Lightweight clients (i.e., laptops)
  - Do not join DHT
  - Only participate in message dissemination
- **Cuckoo-Mobile**
  - Mobile nodes
  - Do not join DHT nor message dissemination

Support for Client Heterogeneity (2)

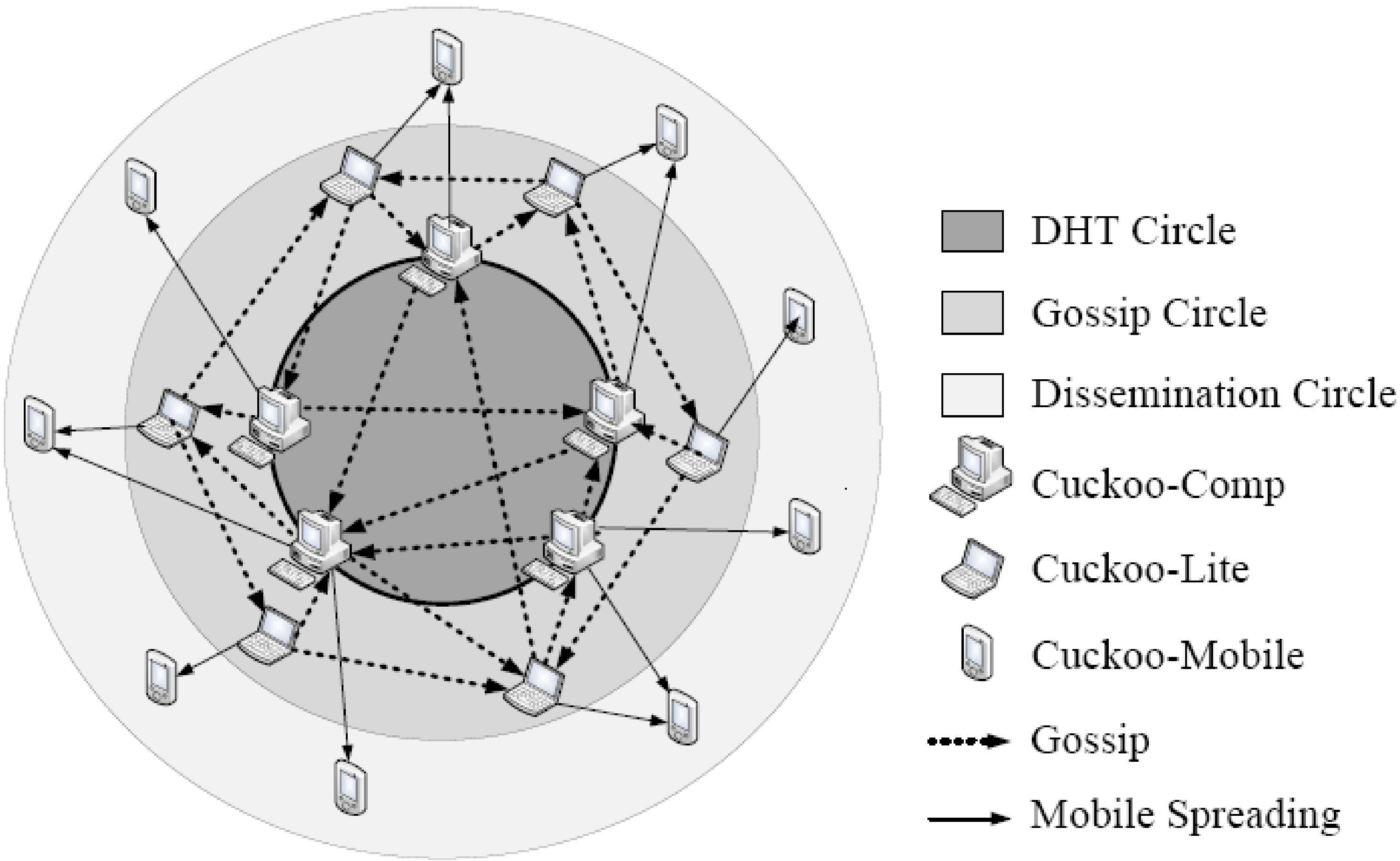


Fig. 11. Content dissemination with heterogenous user clients

Message Loss

Messages may be lost due to the uncertainty of the randomness of gossip.

Exploit *<u>statusId</u>* to check message loss

- **Made up of two parts: userId + sequence number**

userId	Sequence number

statusId

- **Gaps between the sequence number of *statusIds* => message loss**

Security Issues

- Spam that impersonate normal users
- Violating message integrity by altering contents
  - Digital signature
  - Asymmetric key cryptography
- DoS attack and content censorship
  - Distributed nature
  - Communication based on social/media network
  - Mark and upload later
- Brute-force unwanted traffic
  - Trust and reputation model
    - based on social relations

Outline

- Background and Related Work
- Measuring Availability at High Load
- Key Design Rationales of Cuckoo
- **Performance Evaluation**
- Conclusion

Dataset (1)

1. **Use the raw dataset containing 3,117,750 Twitter user traces (profile, social links, tweets)**
2. **Filter out the 1-week part from Feb. 1 to 7, 2010**
3. **Use BFS to get a subset containing 30,000 users**
4. **Prune the irrelevant social links**
5. **Separate social and media with threshold 100**

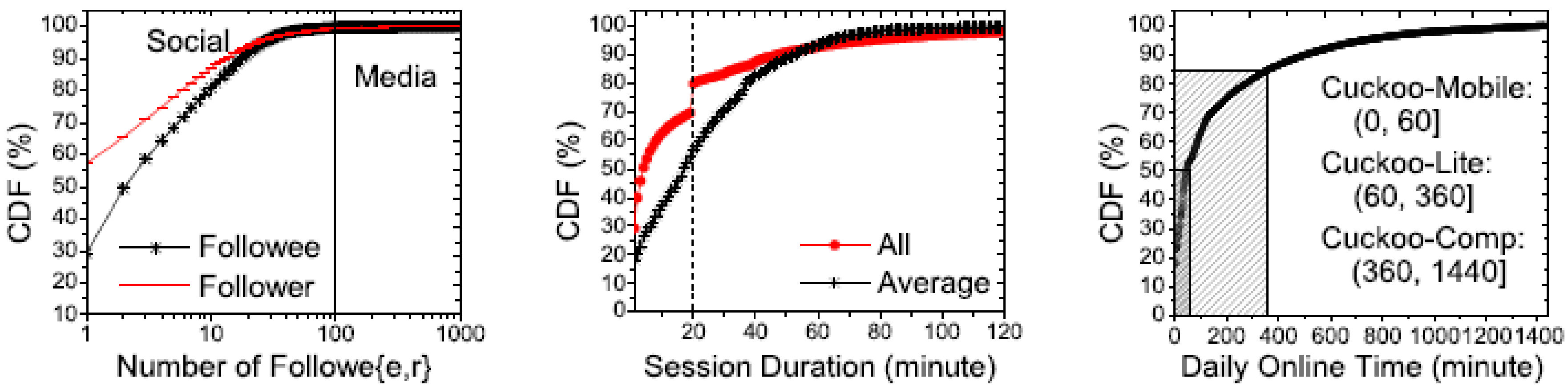


Fig. 6. Dataset: (a) CDF of users' followe{e,r} number; (b) CDF of each session duration; (c) CDF of each users' total online time.

Dataset (2)

6. Use the OSN session duration dataset provided in [2]
7. Classify the three types of Cuckoo users according to their daily online time
   - Cuckoo-Comp: (360, 1440]
   - Cuckoo-Lite: (60, 360]
   - Cuckoo-Mobile: (0, 60]

   ⇒About 50% Cuckoo peers are Cuckoo-Mobile clients.

Thus, we get the one-week dataset which reconstructs the part of Twitter's traffic patterns from Feb. 1 to Feb. 7, 2010.

Implementation

- A prototype of Cuckoo using Java comprises both the Cuckoo peer and the server cloud.
- The prototype of Cuckoo peer
  - 5000 lines of Java code
  - Three types of clients, i.e., Cuckoo-Comp, Cuckoo-Lite, and Cuckoo-Mobile
  - Use *socket* to implement end-to-end message delivery
  - Different types of application layer messages
  - Use XML for local data management
- The prototype of server cloud
  - 1500 lines of Java code
  - Use plain text files to store user information

Deployment

- **Deploy 30,000 Cuckoo peers on 12 machines**
  - Locate these machines into 2 LANs connected by a 10 Gb/s Ethernet cable
  - Run the one-week Twitter trace
- **Deploy 4 servers to build the server cloud**
  - Let the servers share storage
    => no inconsistency problem

Server Cloud Performance
- Resource Usage

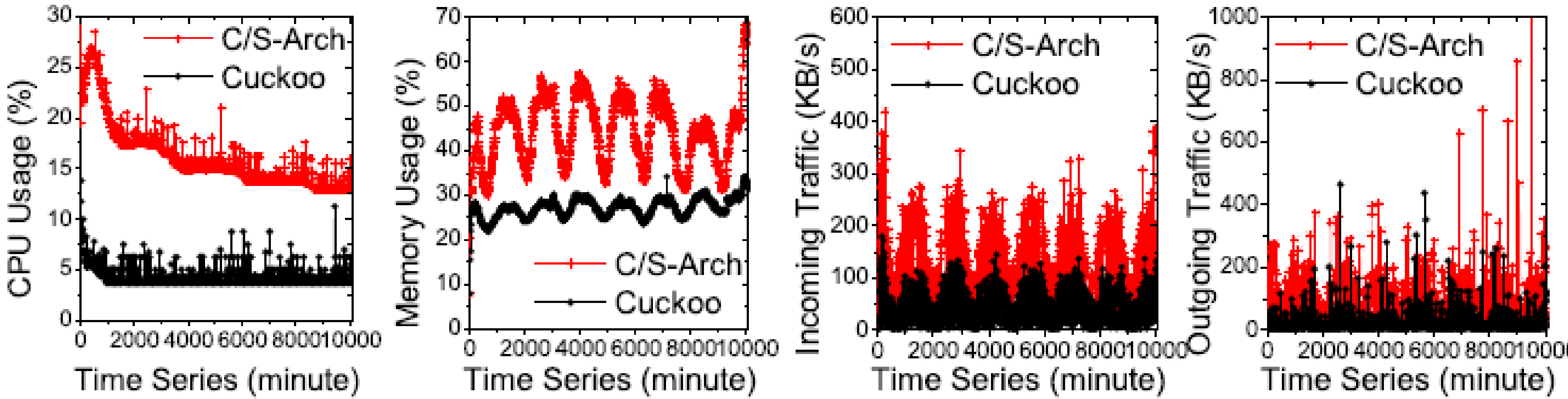


Fig. 7. Resource usage of the server cloud in time series (from Feb. 1 to Feb. 7, 2010): (a) CPU; (b) Memory; (c) Incoming traffic; (d) Outgoing traffic.

❑ Results

1. **50% CPU usage reduction**
2. **50%/16% memory usage reduction at peak/leisure time**
3. **50% bandwidth consumed savings for both incoming and outgoing traffic**

Server Cloud Performance
- Message Overhead & Response Latency

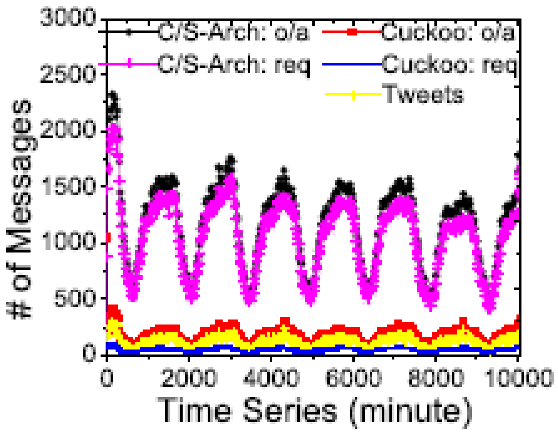


Fig. 8. Message overhead of the server cloud

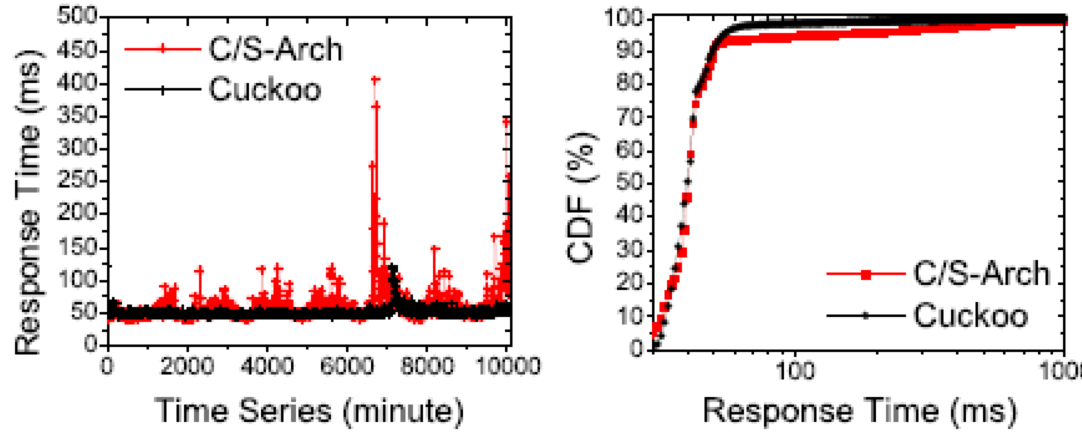


Fig. 9. Response latency of the server cloud: (a) Latency in time series; (b) CDF of the latencies.

❑ Results

1. **80% message overhead reduction**
2. **Smaller peak-valley difference of message overhead (140 vs 1100)**
3. **Requests can be satisfied within 50 ms in most time**

Cuckoo Peer Performance (1)
- Message Sharing

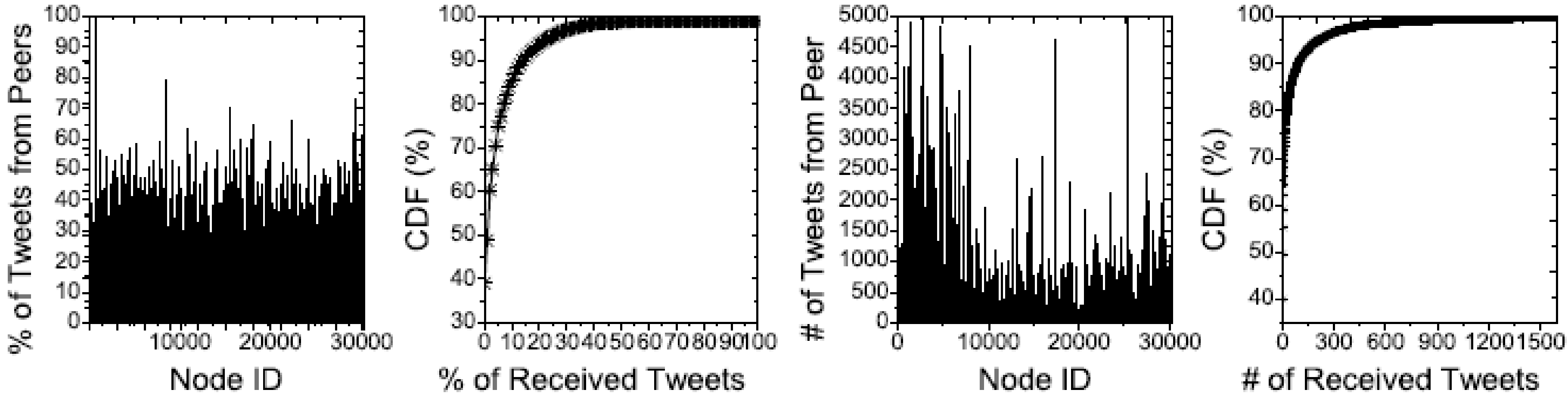


Fig. 10. Message sharing: (a), (b) Percentage and CDF of received tweets against overall tweets; (c), (d) Number and CDF of received tweets.

❑ Results

1. **25+% content exchanging between users**
2. **90% users receive less than 30% subscribed tweets from other peers**
   - The performance is mainly impacted by users‘ access behavior and users‘ online durations
   - The OSN dataset used leads to a pressimistic deviation

Cuckoo Peer Performance (2) - Micronews Dissemination

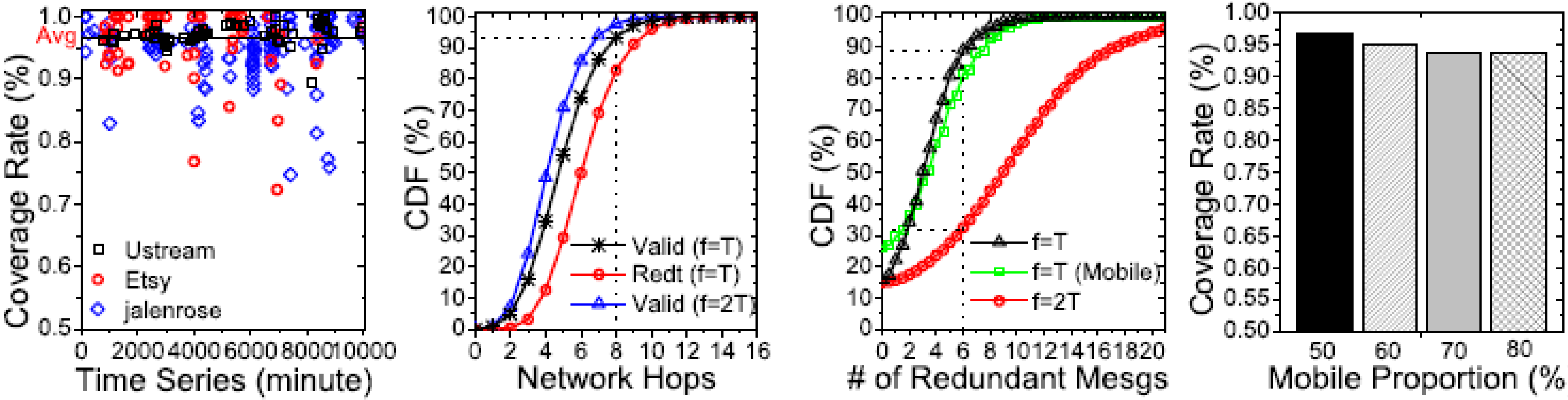


Fig. 11. Micronews dissemination: (a) Coverage; (b) CDF of average network hops; (c) CDF of average redundant messages; (d) Coverage rate with different proportion of mobile nodes.

❑ Results

a) 95+% coverage rate of content dissemination

b) 90% of valid micronews received are within 8 network hops

c) 89% of users receive less than 6 redudant tweets for one dissemination process

d) Stable coverage rate with different proportion of mobile peers

Conclusion

- A novel system architecture tailored for microblogging to address the scalability issues
  - Relieve main server burden
  - Achieve scalable content delivery
  - Decoupling dual functionality components
- A detailed measurement of Twitter
- A prototype implementation and trace-driven emulation over 30,000 Twitter users
  - Notable bandwidth savings
  - Notable CPU and memory reduction
  - Good performance of content delivery/dissemination