Internet Censorship:

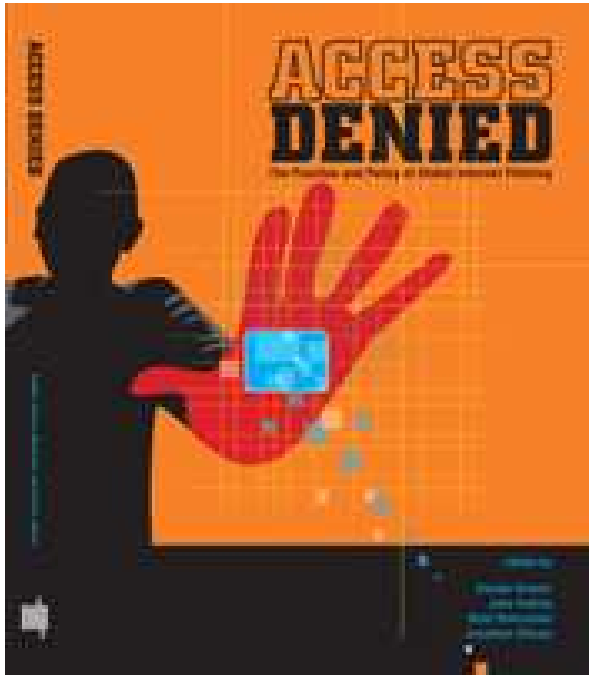
Great Firewall of China (GFC)

Jong C. Park
joumon@gmail.com

Contents

Information throttled

- Censorship observed
  - Slashdot
  - Open Net Initiative
  - Reporters Without Borders
  - Human Right Watch
  - UN, etc.....
- Censorship in burst
  - 3 out of 41 in ‘02 -> more than 25 in ‘09
  - can view at ONI for more details
  - political, social, security/confilict, Internet tools



Generic Filtering

- Maximillian Dornseif, Germany,
  - *from "Government mandated blocking of foreign Web content"*
- Packet-level filtering
  - mostly done by firewall (rules), but need up-to-date
  - OSI layer 3 filtering
    - inspects header and forwards/drops
    - coarse-grained due to high false positive (overblocking)
  - OSI layer 4 filtering: finer-grained than layer 3

Generic Filtering (cont'd)

- Application-level filtering
  - inspects payload and performs most detailed
  - often provides ways of informing users about the filtering
  - hard to be done in real-time due to overhead
  - encryption/compression makes infeasible

Generic Filtering (cont'd)

- Filtering mechanisms
  - proxy firewall (HW requirement, latency)
  - IDS (signature-based and/or heuristic)
  - DNS poisoning
  - help from search engines
  - web-site delisting
  - keyword filtering
  - keyword forgery

Circumvent Censorship

- Discard TCP RST
- Encrypt/Compress
- Use proxy server (web cache)
- Use publicly accessible DNS server (caveat: security)
- Tunneling such as ICMP, SSH, SSL, or VPN
- Add comment in HTML
- Use captchas
- etc

Great Firewall of China

- Why China?
  - Most complicated censorship mechanism
  - China filters to maintain power
  - Cyber-sensors and cyber-police
  - Hierarchical supervisory bodies
  - World’s largest IPv6 backbone

GFC (cont'd)

- A large number of sites block
- Not all about pornography sites
  - PA state, USA, blocks child pornography
- Most advanced keyword filtering
  - falun gong, human right, Taiwan, Tibet, even foreign cities due to similar pronunciations
  - TCP RSTs, specious SYN/ACK

Zittrain & Edelman

- First studied on blocked web sites in China
- *from "Internet filtering in China", IEEE Internet Computing*
- Studied filtering methods
  - IP address blacklisting
  - DNS IP address blocking
  - DNS redirection
  - URL keyword filtering
  - HTML response keyword filtering

Ignore the GFC

- Seminal work by Richard Clayton et al.
  - *from "Ignoring the Great Firewall of China", 6th workshop on Privacy Enhancing Technologies*
- Filtered at borders
- Triggers 3 consecutive TCP RSTs (seq, seq+1460, seq+4380)
- Stateless inspection

ConceptDoppler

- More work done by Jed Crandall et al.
  - *from "ConceptDoppler: a weather tracker for Internet censorship", ACM CCS, 2007*
- Showed GFC as panopticon
  - used tcptraceroute with increasing TTL
- Also showed 28.3% requests survived to reach server
- Most filtering happens at 1st hop, but deep thru 13th hops beyond borders
- Provides blacklist words using LSA

TCP RST patterns

- Work done by N. Weaver et al.
  - *from "Detecting forged TCP RST Packets", TRUST, 2008*
- Studied on forged TCP RST patterns and fingerprints
- Patterns:
  - D+R, R+D, R+R, S+R, SA+R
- Fingerprints:
  - China has IPID 64, IPID -26, SEQ 1460, etc

Electronic Big Brother

- Work done by J. Karlin et al.
  - *from "Nation-state routing: censorship, wiretapping, and BGP"*
- Countries with most information flow: USA, England, and Germany
  - China has little effect on interdomain routing
- Power Law can be applied

Our Work being going…

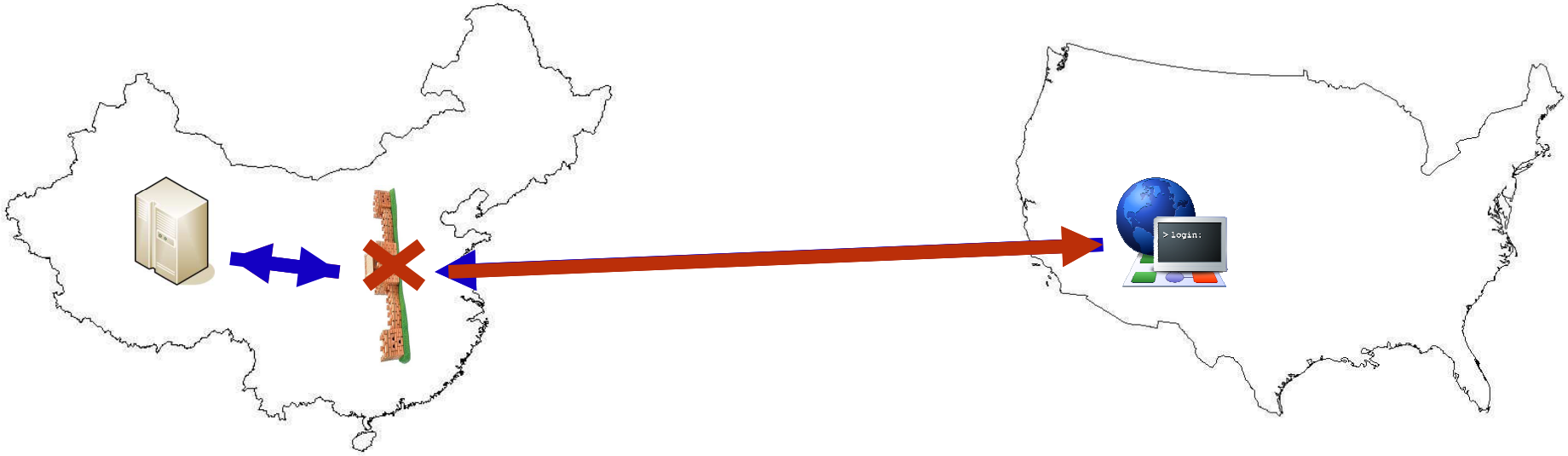
- J. Crandall et al. showed, with a blacklist, that GFCs blocks HTML GET requests substantially
- Are GFCs symmetric or asymmetric in HTML filtering?
  - If asymmetric, filtering would be less effective
  - How much blocking on HTML response?
  - What blacklist keywords blocked & ***why***?
  - How do GFCs block & how to escape?

Constraints

- Internet Measurement
  - No fine-grained protocol support
  - Privacy and legal issue
  - and more…
- China claims world's largest IPv6 backbone
  - Huge latency due to 4over6
  - Little collaboration from IPv6

Keyword Probe

GET server/keyword=*falun* HTTP/1.1



Doesn’t know which direction is blocked…

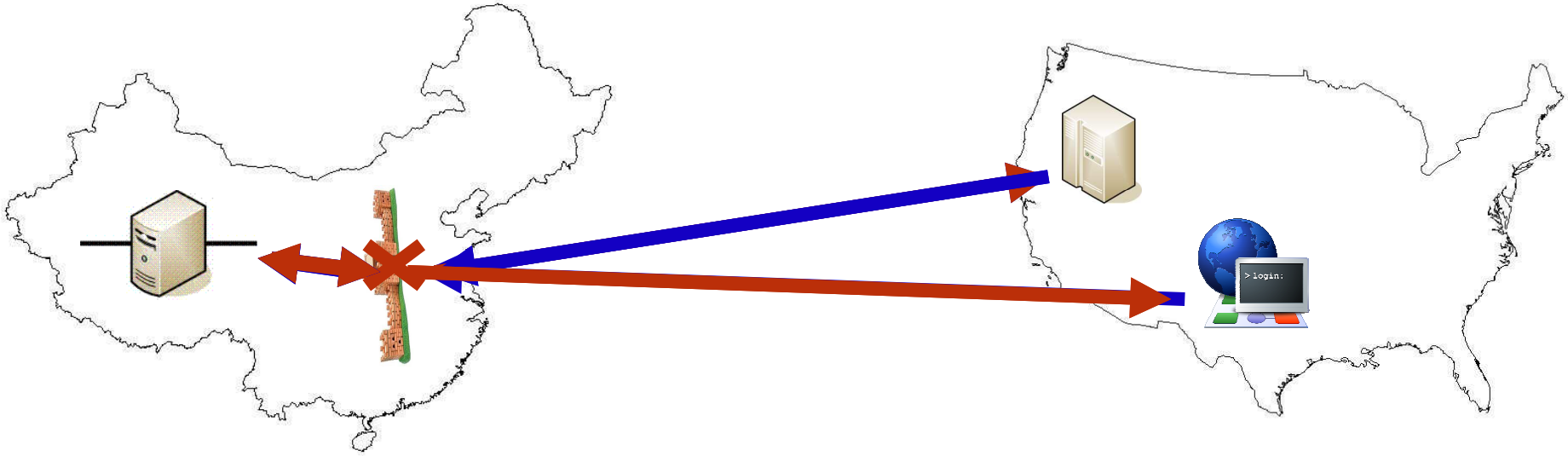
Number Probe

- Use a proxy server, a.k.a. a web cache
- Use number query instead of keyword
  - Chinese cannot be guaranteed
  - Neither can English since some abbreviations are blocked
  - Bogus response page of ~4KB
  - Hash will be much better, but file used currently
    - Latency generated due to table iteration

Number Probe

GET server/keyword=*1* HTTP/1.1

1 means falun



Doesn’t matter which direction is blocked…

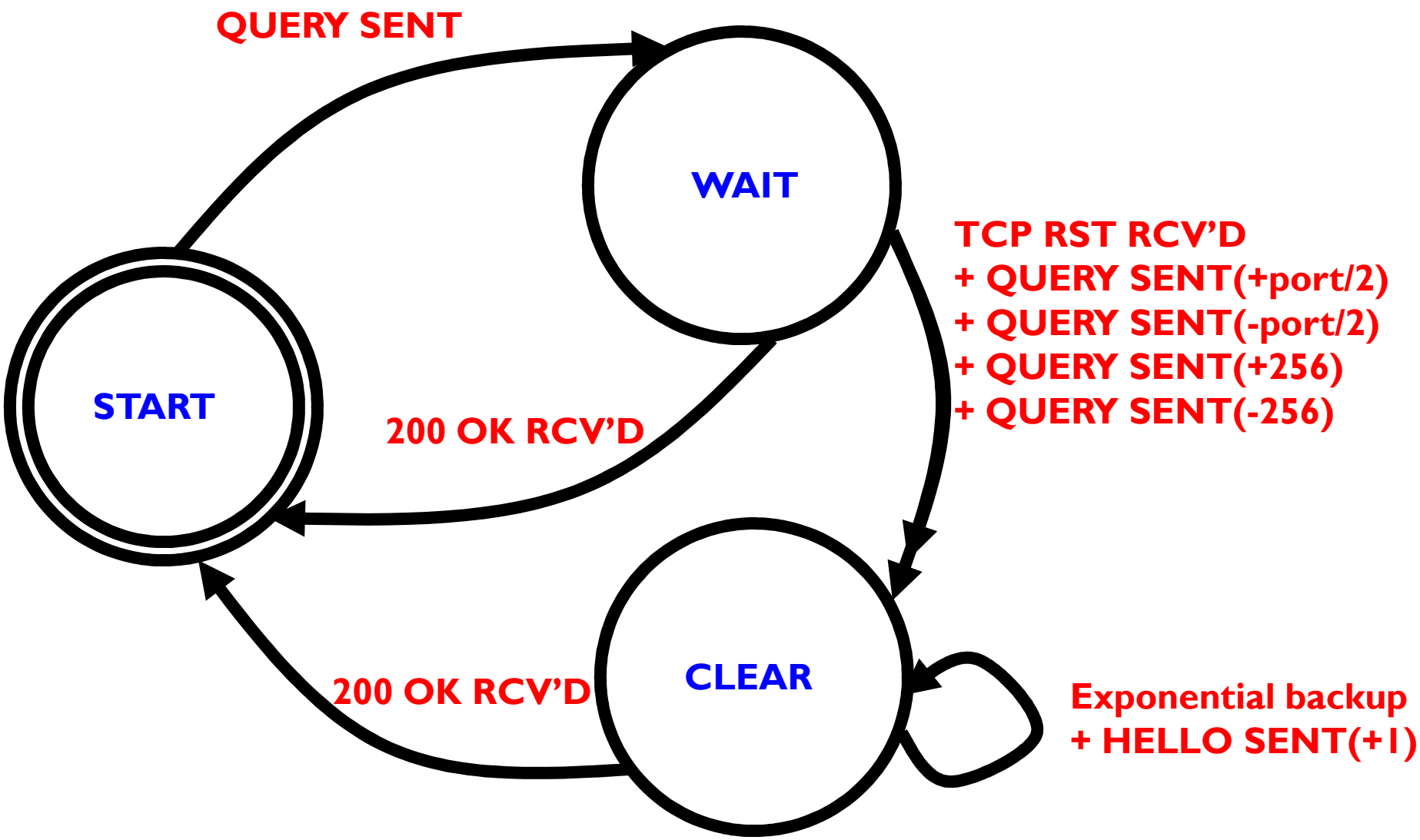
Setup

- 7 pairs of single server and single client
- Scripting probes with wget command + Python code on server
  - WGETRC, .wgetrc, or /etc/wgetrc
  - Scapy used

Setup

- Python code with libpcap record traffics on both server-side & client-side
  - tcptraceroute recording latency every 10 mins
- 3 contiguous probes sent
- Preliminary tests from late Feb to early Mar
- Real experiments from spring recess thru now

FSM for Probe



Proxy Hunt

- Chinese broadcast a list of proxy servers
  - Financial issue
  - Falun movement
- Geographically diverse web caches
  - Use Visualroute server 2008 trial
    - Returns ***guess locations*** from ISP info
- Proxy life time
  - Last up to two weeks or so
  - Some lasts a couple of days

What to test?

- 12/20 diverse locations randomly chosen
  - Most along with east coast (1 partial)
  - 1 inland location included, but partial
  - 1/2 hrs to 2 days each run
- Different locations of a single keyword
  - Beginning/Middle/End
- Different response sizes
  - 600B/4KB/40KB/175KB/350KB
- Static vs dynamic pages

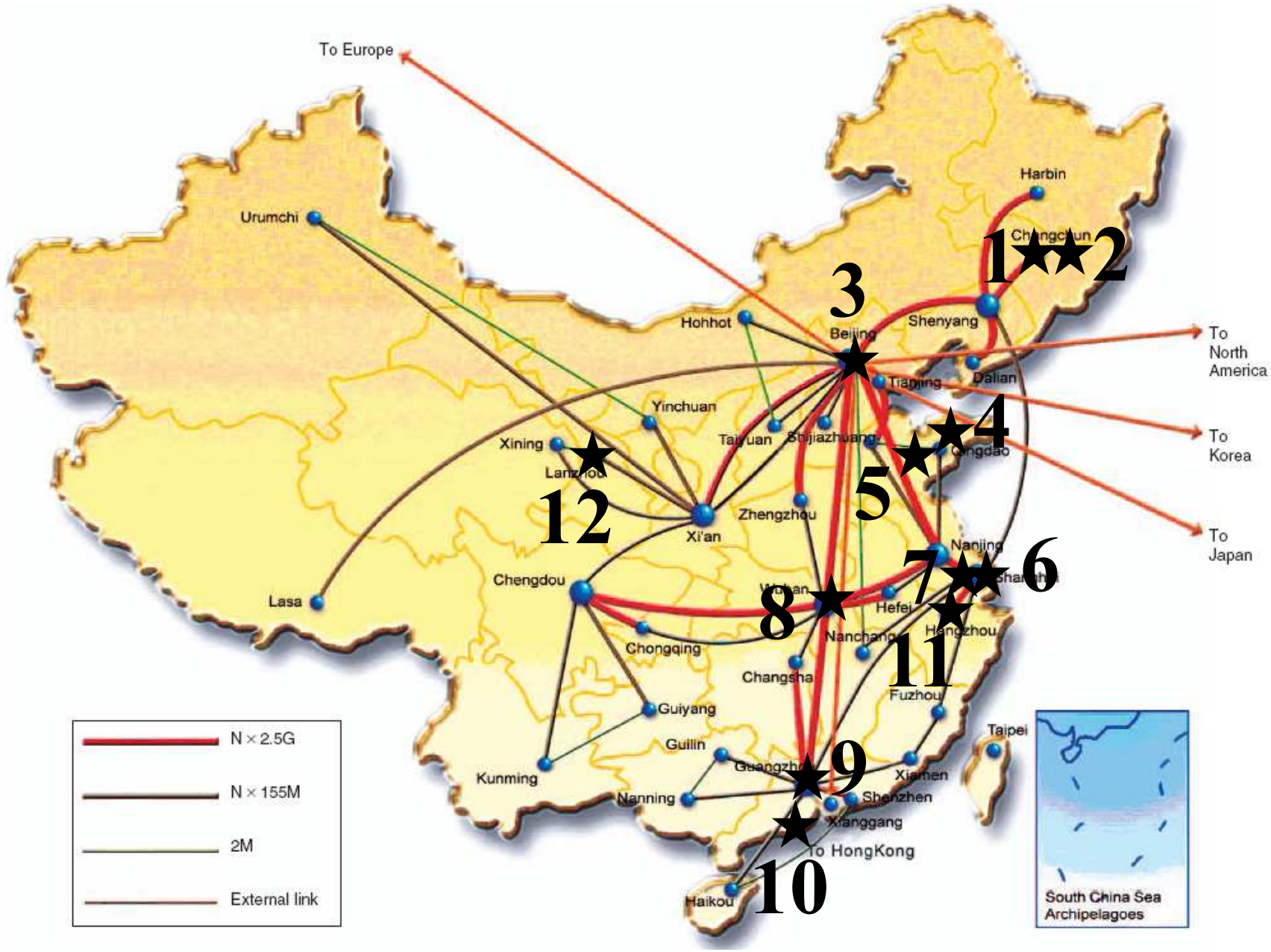
What to test?

- Keyword threshold (***to be tested***)
  - How many number of the same/different kind triggers & how?
- Different HTTP protocols (***to be tested***)
  - Probably no effect since proxy uses HTTP 1.1
- No STDDEV data available yet (***being tested***)
- TCP RSTs trend: how many & how (***to be tested***)
- Any other ideas?

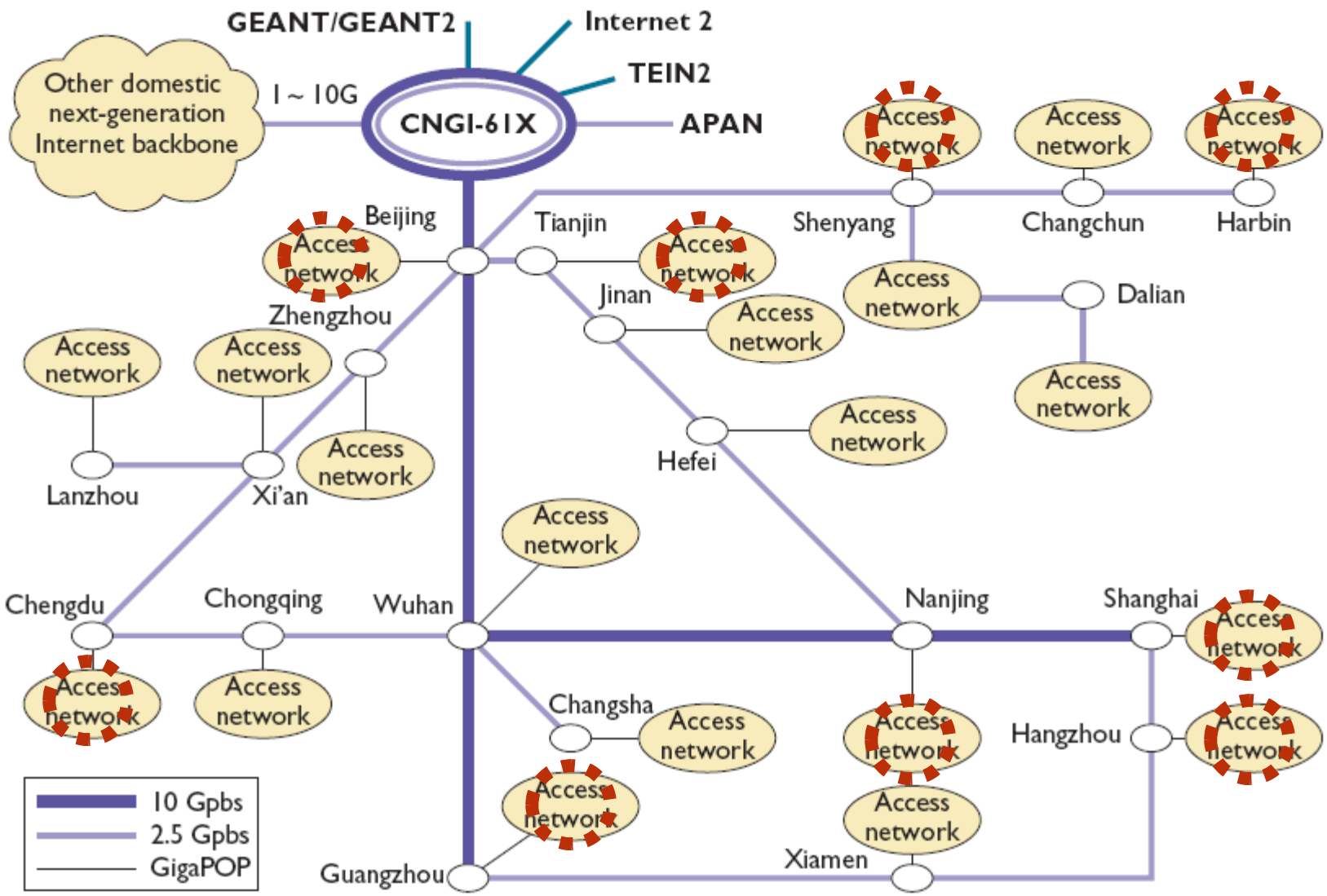
Map


Mongolia
Ulaanbaatar
Wulumuqi
乌鲁木齐市
Bishkek
Kyrgyzstan
Changchun
长春市
Huhehaote
呼和浩特市
Yinchuan
银川
Xining
西宁
Taiyuan
太原市
Jinan
济南市
North Korea
South Korea
中国
China
Xi'an
西安市
Zhengzhou
郑州市
Wuhan
武汉市
Hefei
合肥市
Shanghai
上海市
Hangzhou
杭州市
Chengdu
成都市
Chongqing
重庆市
Lhasa
拉萨市
Changsha
长沙市
Guiyang
贵阳市
Fuzhou
福州市
Kunming
昆明市
Nanning
南宁市
Hong Kong
香港
Macau
澳門
Taiwan
台灣
Hà Nội
Hải Phòng
Haikou
海口市
Laos
Myanmar
(Burma)
Bangladesh
Daulatpur
Kolkata
India
Nepal
Bhutan
Peshawar
Islamabad
Amritsar
Faisalabad
New Delhi
Agra
Jaipur
Jodhpur
Kanpur
Patna
Varanasi
Bhopal
Indore
Nagpur
Surat
Mumbai
Hyderabad
Fukuoka
福岡
Osaka
大阪
Hiroshima
広島
Kumamoto
熊本
Miyazaki
宮崎
Naha
那覇
1
2
3
4
5
6
7
8
9
10
11
12

IPv 6 Backbone



IPv 6 Backbone



TCP RST

- Depending on location & probe, the number of TCP RSTs received vary
  - Sometimes just 1 RST
  - Sometimes more than 10 RSTs
  - No automatism at this time (***future work***)

Odd Responses

- Unlike HTML GET request, HTML response receives odd packets
  - 404 Page Not Found (once)
  - 502 Error (58 times)
    - Proxy error/Bad gateway
  - 503 Service Unavailable (236 times)
  - Connection Timed Out (***not counted yet***): discarded

Odd Responses

- Two kinds
  - Just 502/503 error received without any preceding/following RSTs
    - Some HELLOs get TCP RSTs (delay)
  - RST received first and then 502/503 error received: categorized same as TCP RST
- Who sends these? Let’s look at client-side!

200 OK

When a non-blocked keyword is requested

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0
HTTP/1.0, TTL=64 ** 0 means hello*
China -> UNM: [ACK] seq=1, ack=138, TTL=48
China -> UNM: [TCP Dup ACK] [ACK] seq=1, ack=138, TTL=48
China -> UNM: [TCP segment of a reassembled PDU] TTL=48
China -> UNM: [TCP segment of a reassembled PDU] TTL=48
China -> UNM: [TCP previous segment lost] [FIN+ACK]
seq=4262, ack=138, TTL=48
UNM -> China: [ACK] seq=138, ack=2897, TTL=64
UNM -> China: [TCP Dup ACK] [ACK] seq=138, ack=2897
China -> UNM: [TCP out-of-order] HTTP/1.0 200 OK
(text/html)

TCP RST

When a TCP RST packet is received

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0
HTTP/1.0, TTL=64 ** 0 means falun*
China -> UNM: [ACK] seq=1, ack=152, TTL=48
China -> UNM: [RST] seq=1, TTL=50
UNM -> China: [FIN+ACK] seq=1, ack=1, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64

502,503 Error

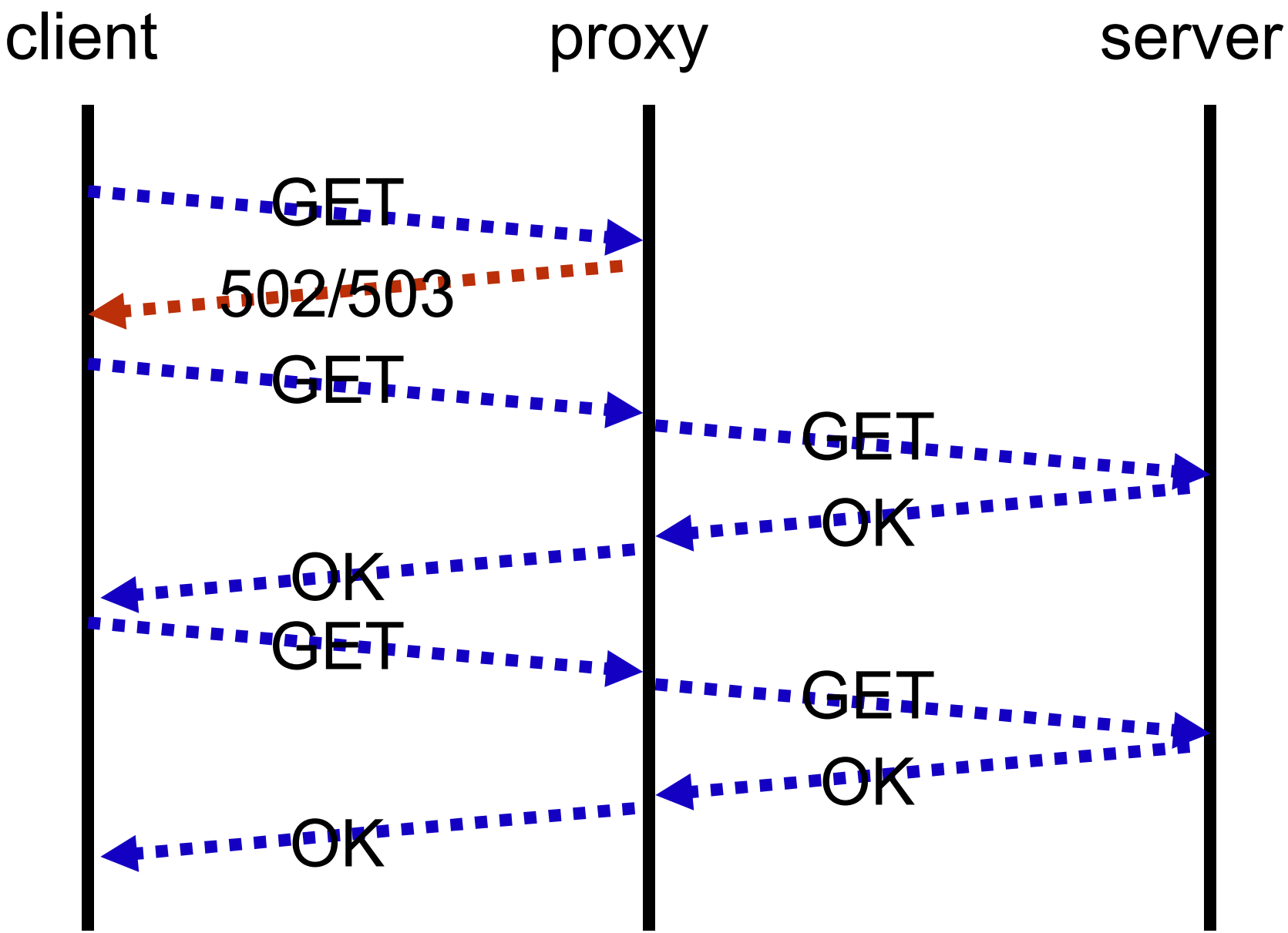
When a 503 error message is received

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0
HTTP/1.0, TTL=64 ** 0 means falun*
China -> UNM: [ACK] seq=1, ack=134, TTL=48
China -> UNM: [TCP previous segment lost] [FIN+ACK]
seq=1578, ack=134, TTL=48
UNM -> China: [TCP Dup ACK] [ACK] seq=134, ack=1, TTL=64
China -> UNM: [TCP retransmission] [TCP segment of
a reassembled PDU] , TTL=48
China -> UNM: [TCP retransmission] HTTP/1.0 503 Service
Unavailable, TTL=48

502,503 Error

- Based on client-side, GFCs seem to intervene?
  - Probably yes or no depending on preceding/following RSTs
- Let’s look at server-side!

502,503 Error



502,503 Error

- Probes right before the current probe and right after the current were successful
  - i-th keyword (3rd probe): OK
  - (i+1)-th keyword (1st probe): 502 Error
  - (i+1)-th keyword (2nd probe): OK

502,503 Error

- What caused this kind of 502/503 error?
  - Probably proxy misconfigurations
  - More analysis needed
  - Some HELLOs get delayed RSTs
- 503 same as 502?
  - No concrete evidence
  - Still in analysis

Odd Packets

- Odd packets are generated by a web cache
  - Mistakenly thought as TCP RST packets received from GFC on the way from server to a web cache
  - Then, self-censorship? Definitely no!
  - Probably due to proxy misconfigurations
- Odd packets can also be considered TCP RST? Some are definitely ***false positive***, though.
  - Why differs? ***Can't explain at this time***

Self-censorship?

When a self-censorship is observed at Beijing

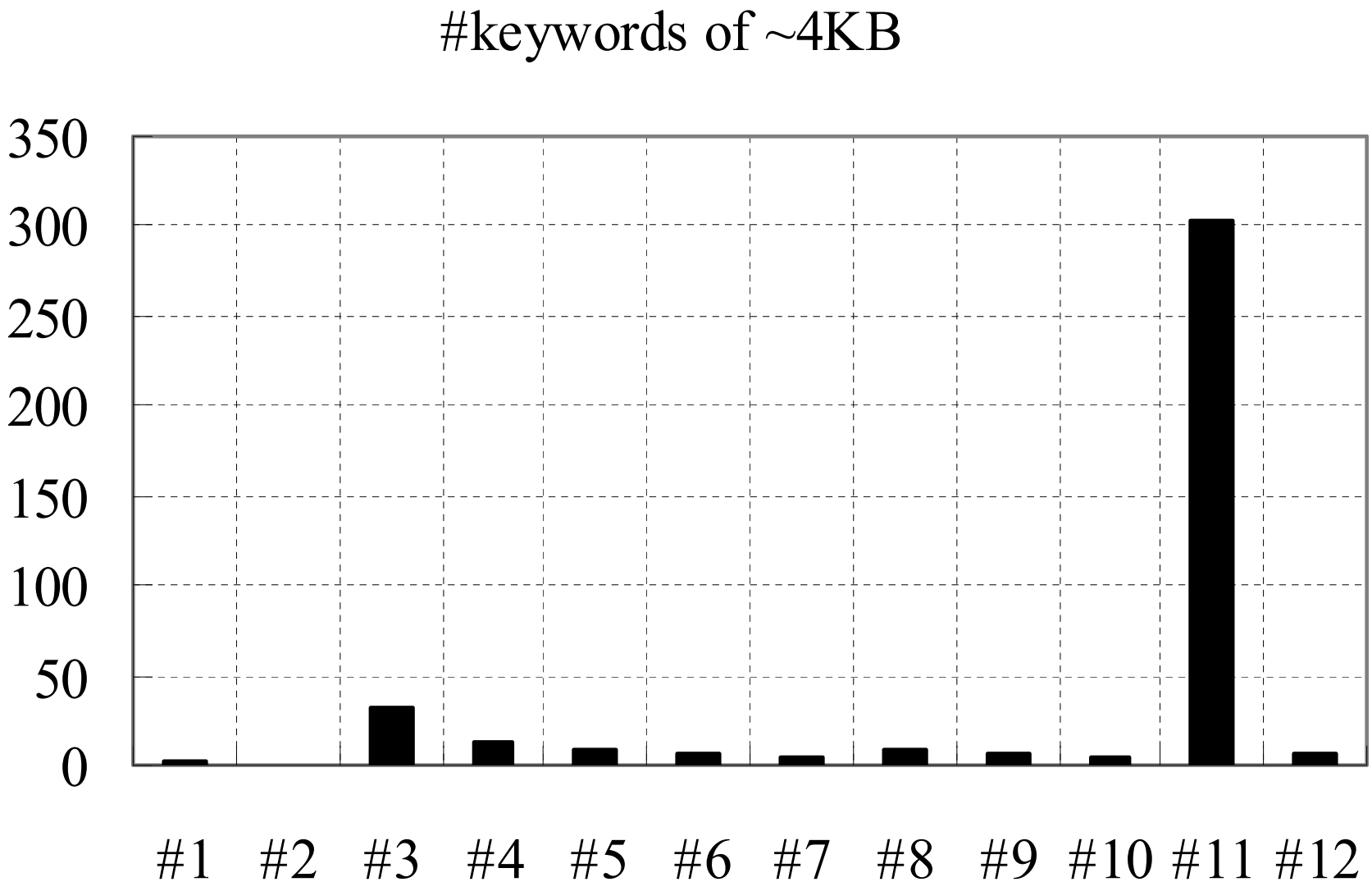
UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=46
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0 HTTP/1.0, TTL=64 **** 0 means falun***
China -> UNM: [ACK] seq=1, ack=152, TTL=46
China -> UNM: [RST] seq=1, TTL=46
UNM -> China: [FIN+ACK] seq=1, ack=1, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64

Proxy ACK'ed server in which TCP RST is not observed at all.

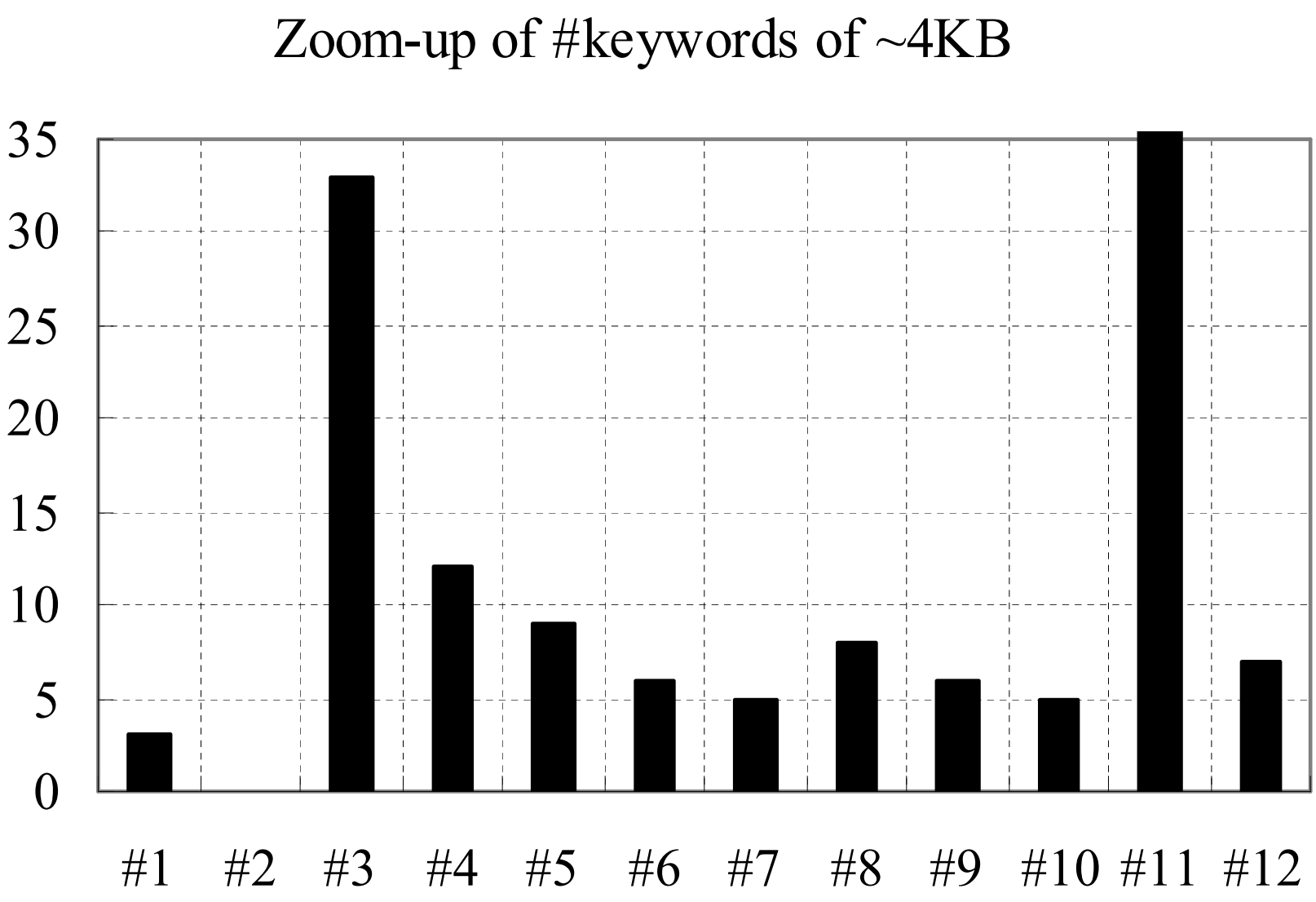
Probing Results

- Odd probing results, but the following should be taken into an account:
  - Route flutter
  - HTML request blocking trend varies depending on probing time (diurnal pattern)
  - Might be the most efficient way because of distributed filtering on response?
- Can’t explain many phenomenon, but can make good guesses at least

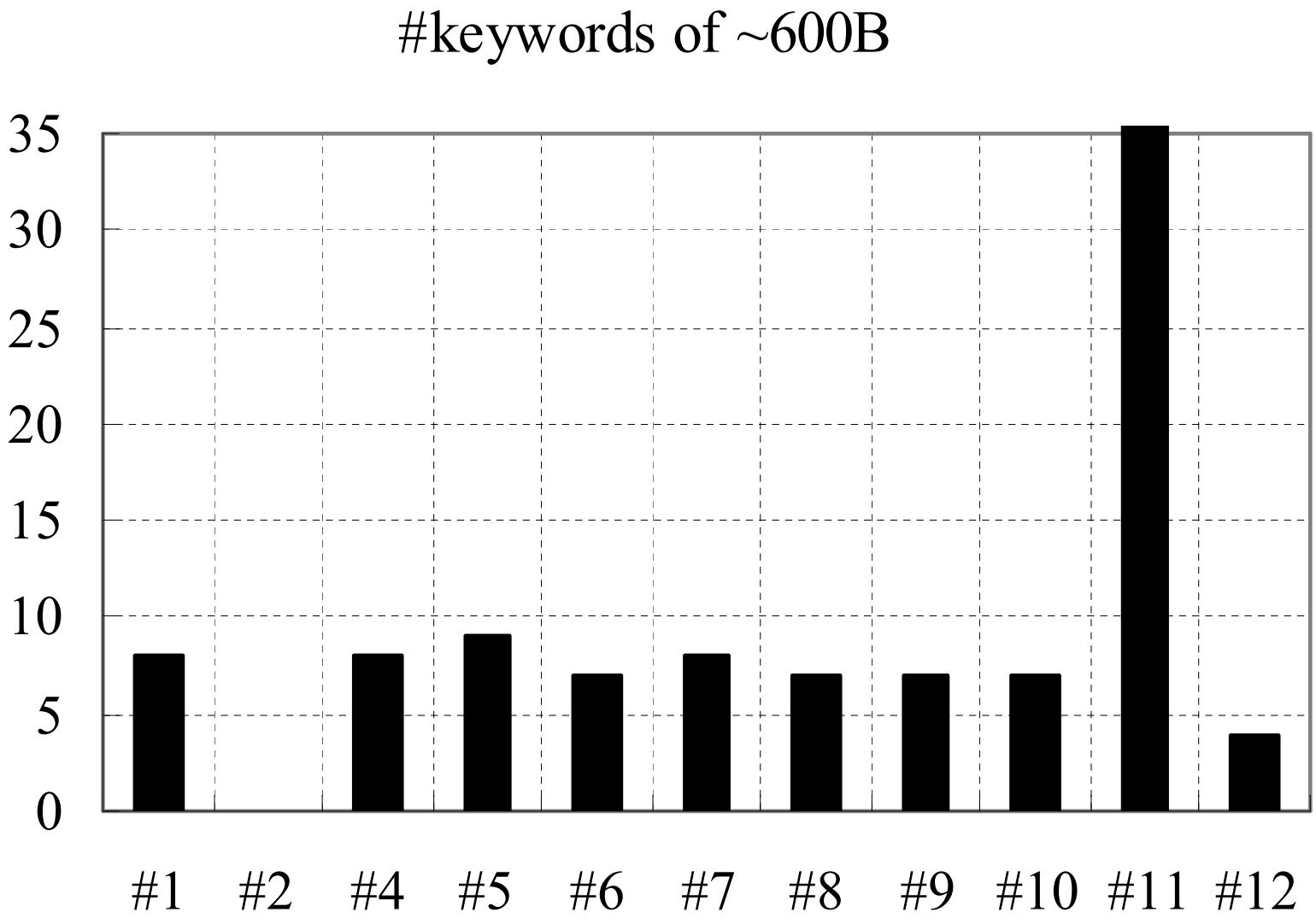
#Block on 4KB



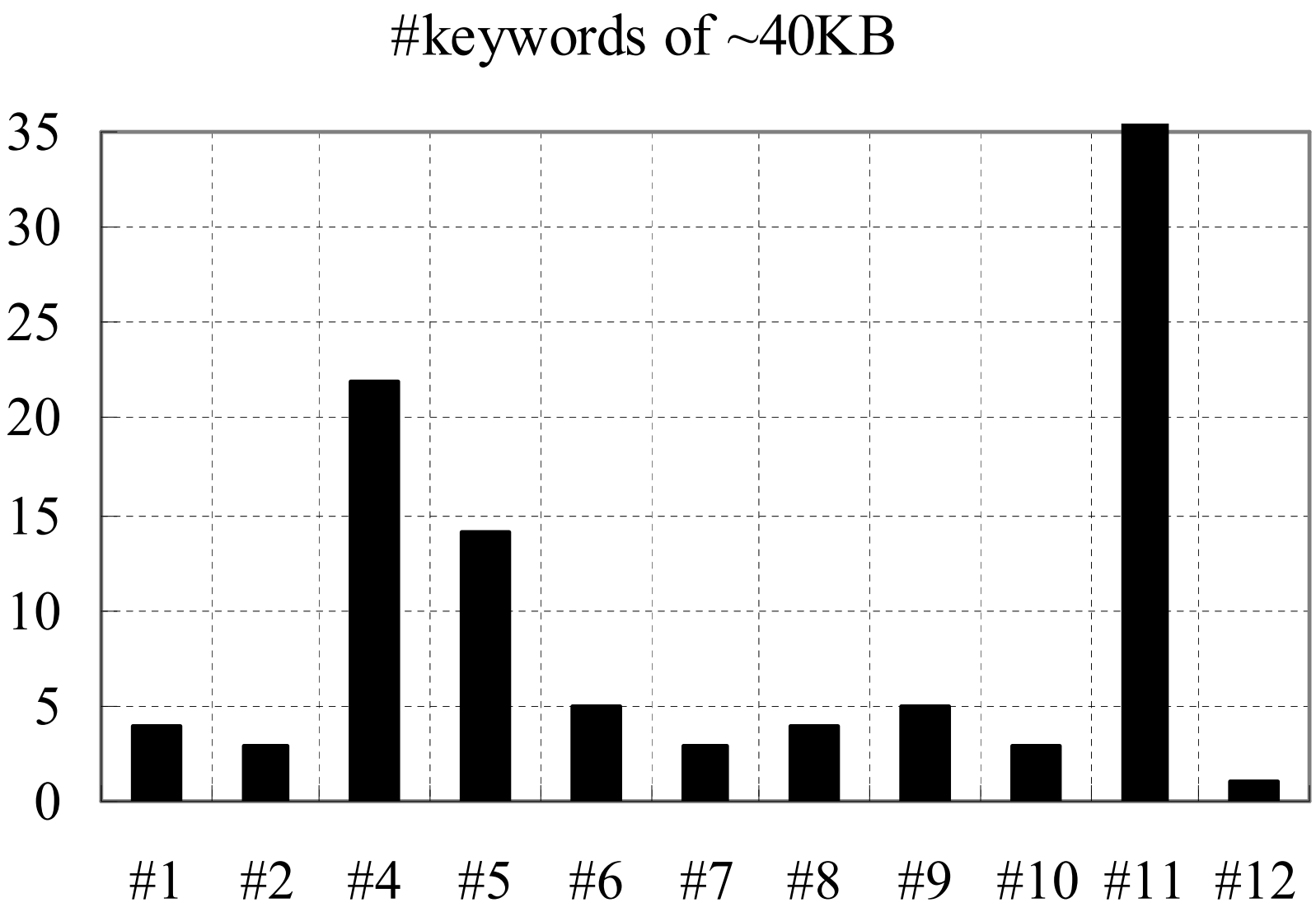
#Block on 4KB


Zoom-up of #keywords of ~4KB
35
30
25
20
15
10
5
0
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12

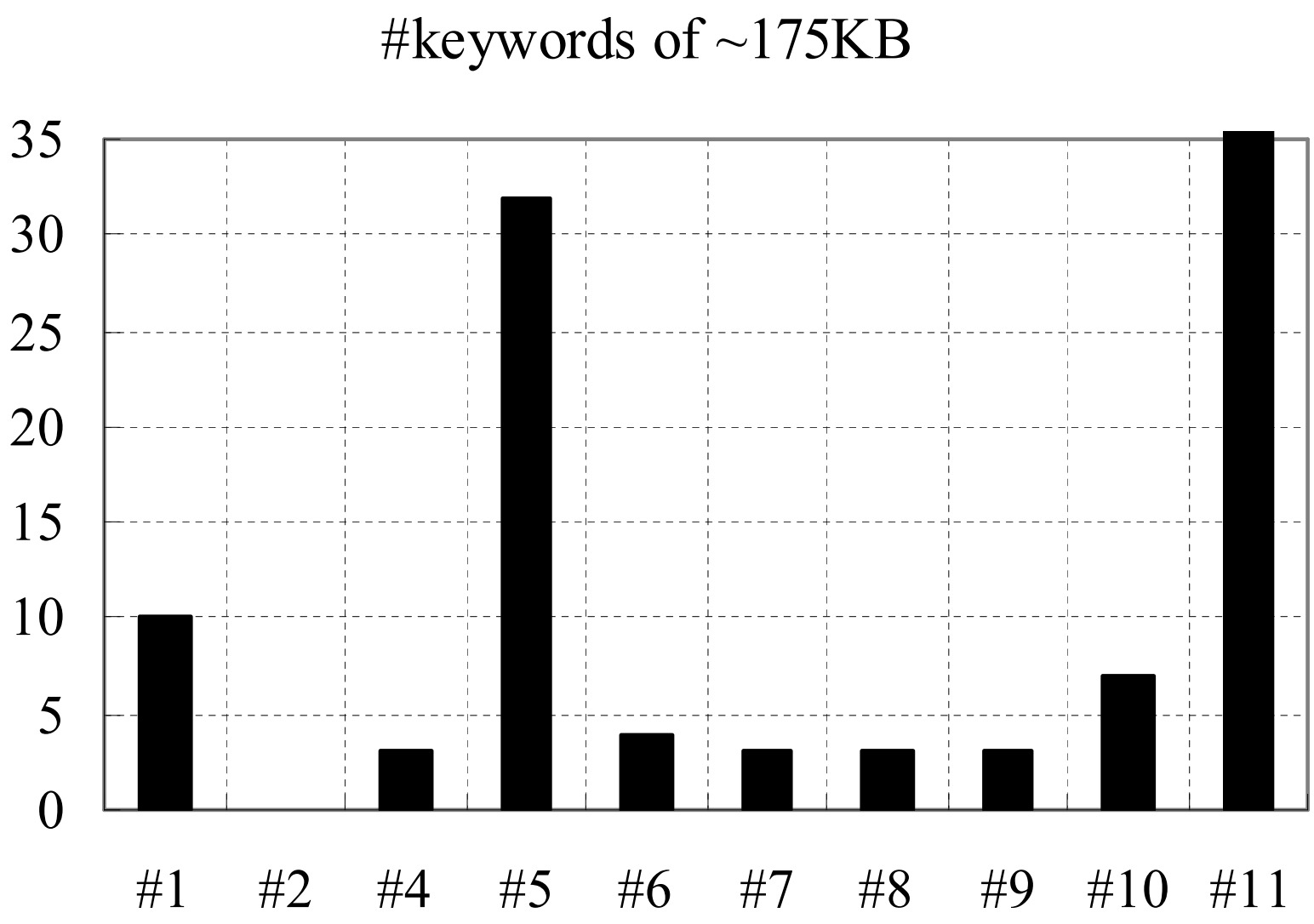
#Block on 600B



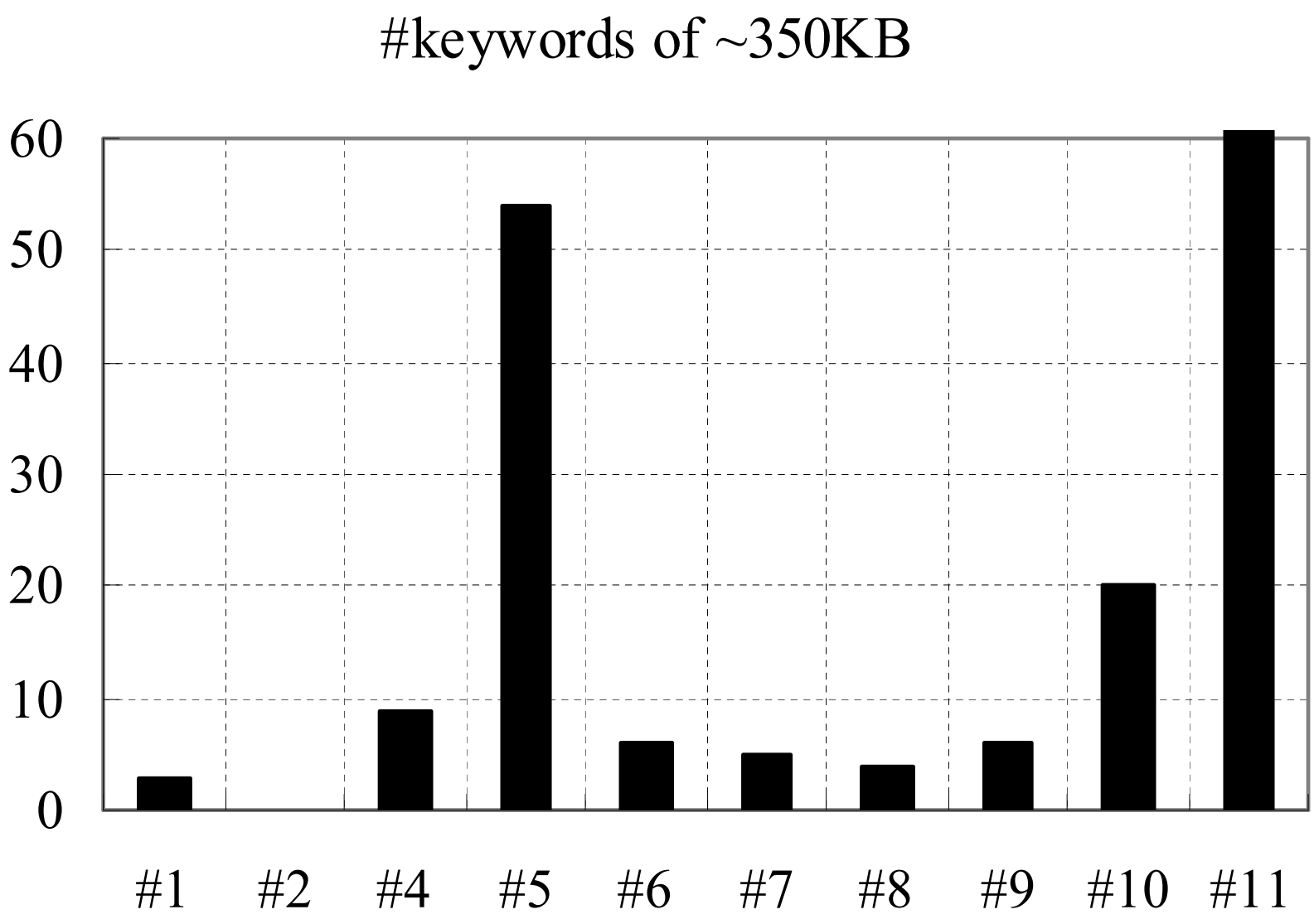
#Block on 40KB



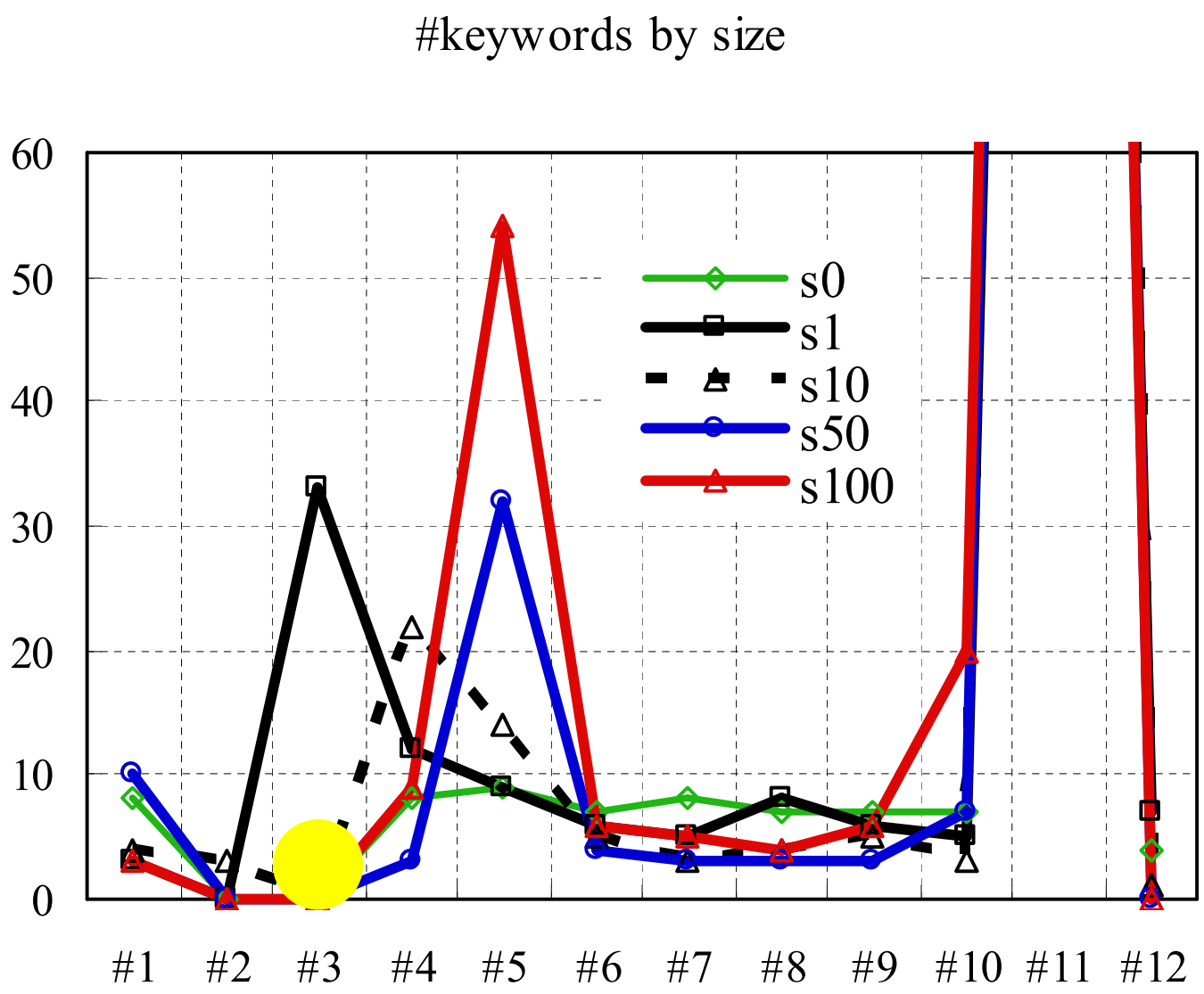
#Block on 175KB



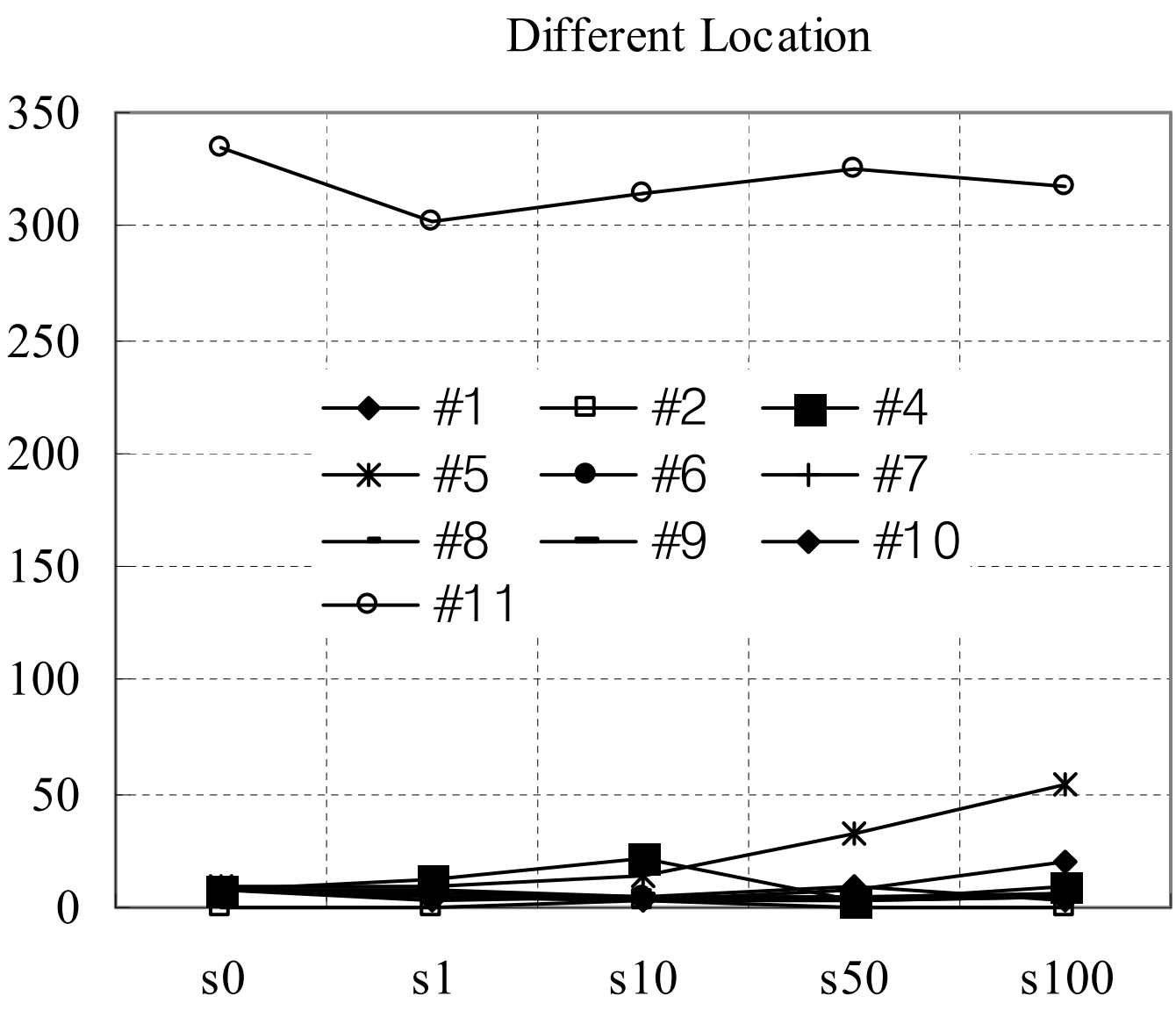
#Block on 350KB



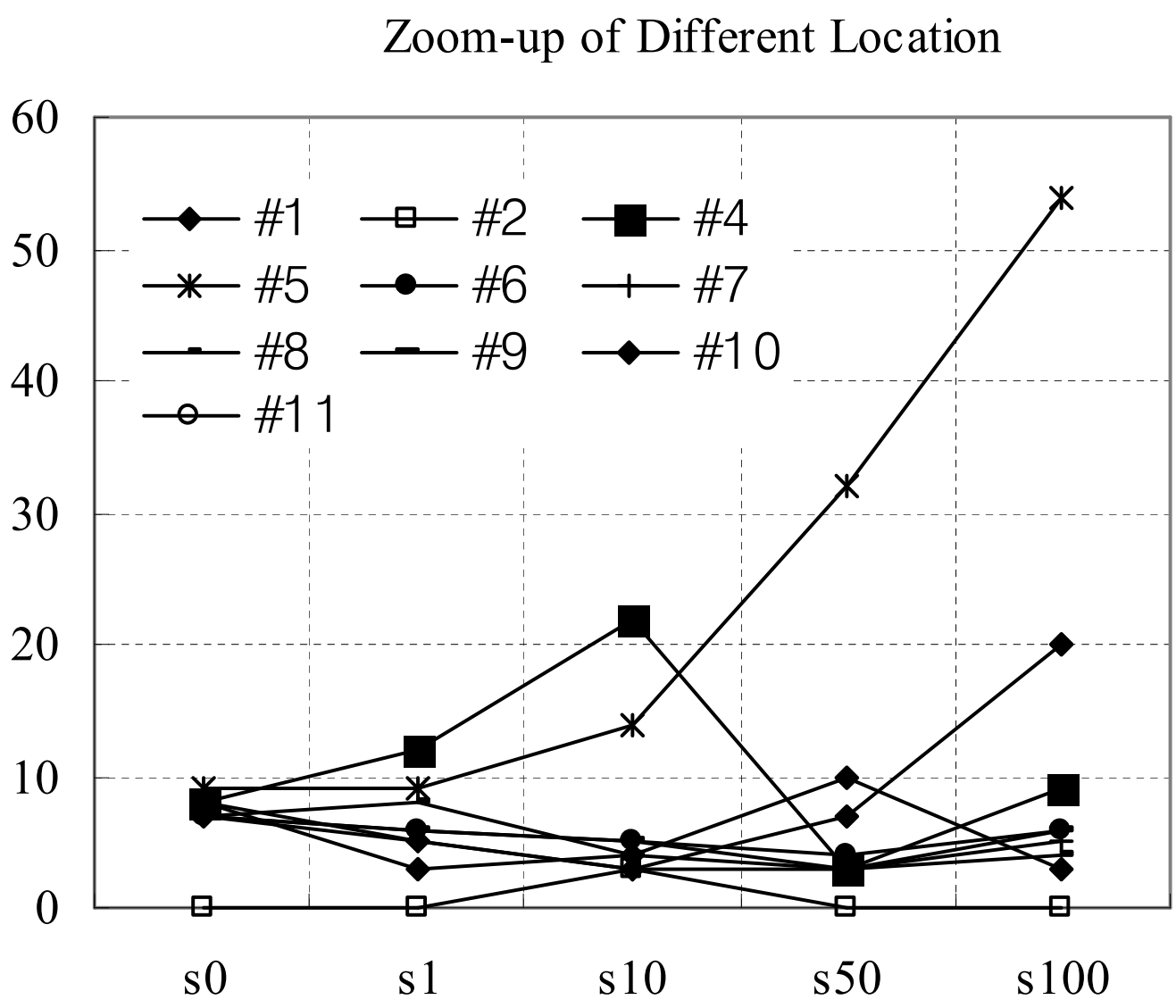
#Block on different size



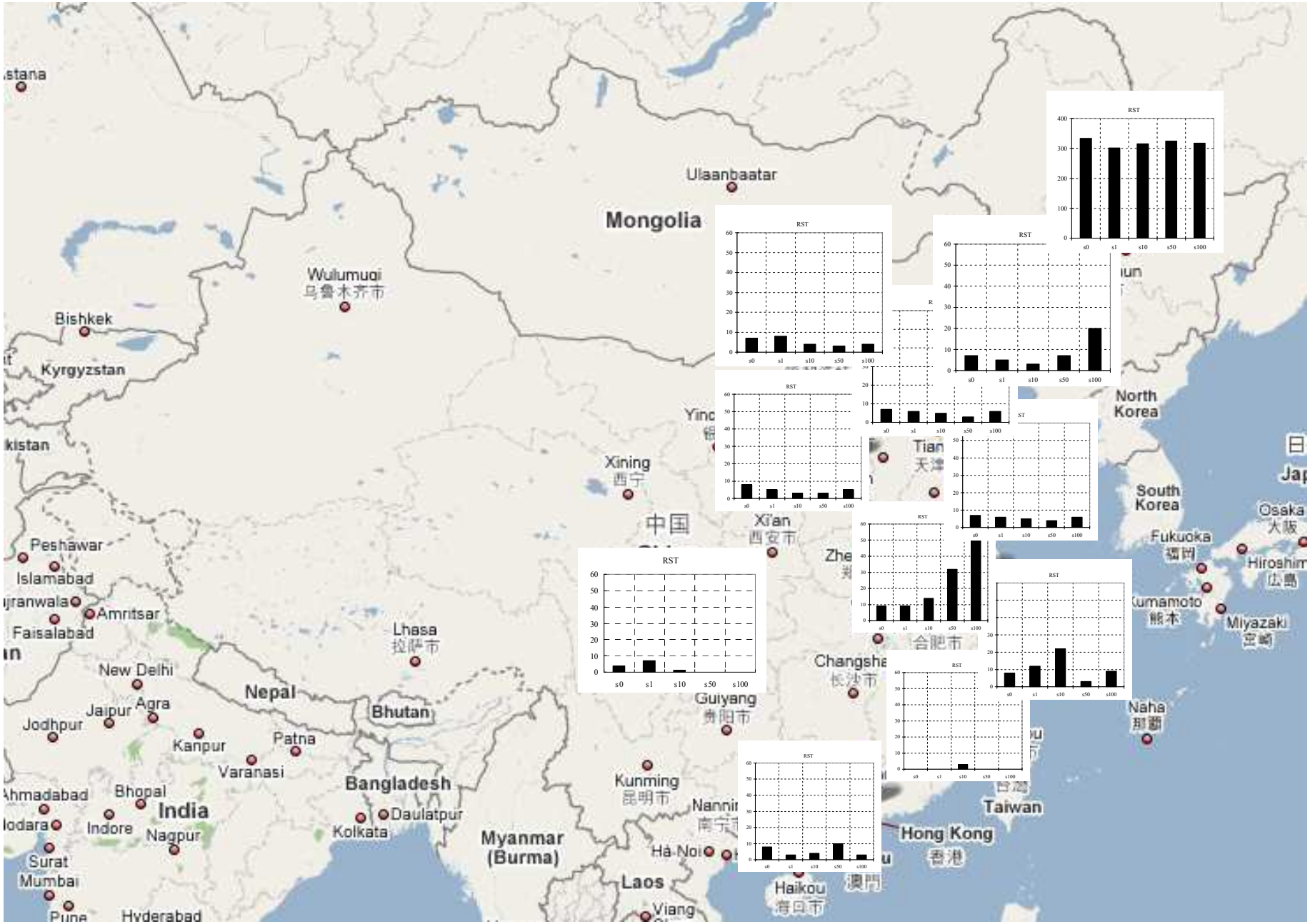
#Block on different loc



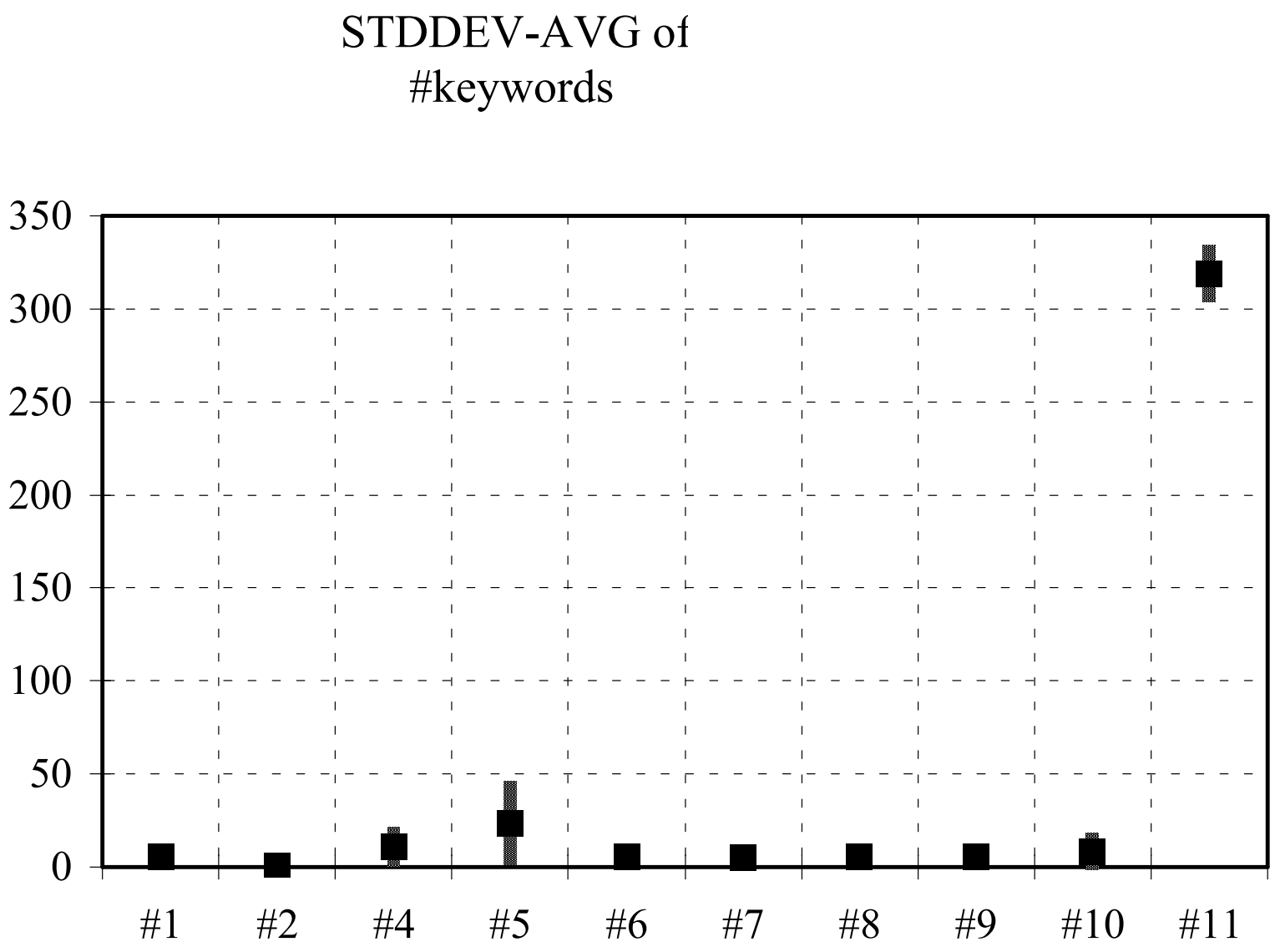
#Block on different loc



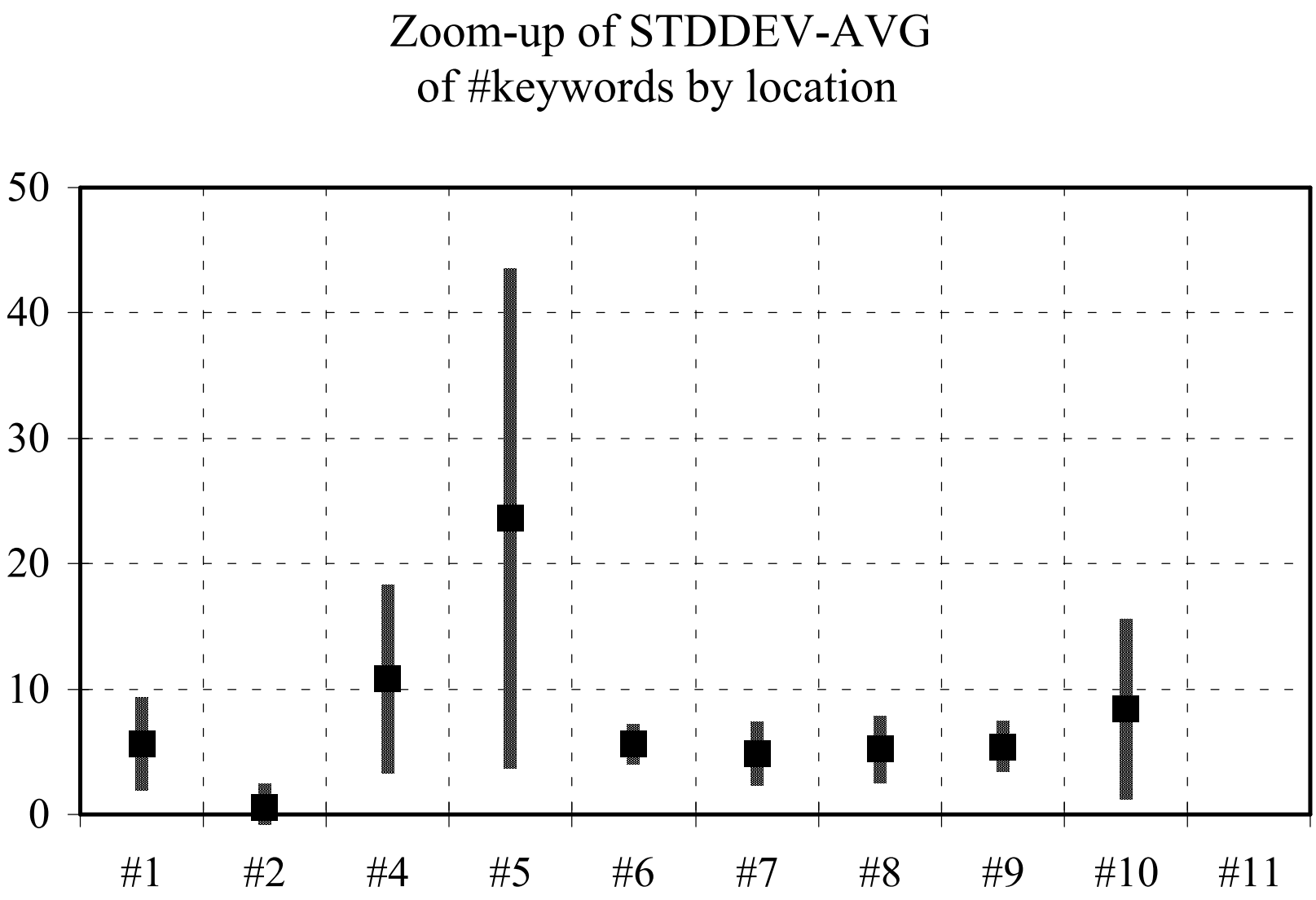
Nation-wide


Ulaanbaatar
Mongolia
Wulumuqi
乌鲁木齐市
Bishkek
Kyrgyzstan
Xining
西宁
中国
Xi'an
西安市
Lhasa
拉萨市
Peshawar
Islamabad
Amritsar
Faisalabad
New Delhi
Nepal
Bhutan
Jaipur
Agra
Jodhpur
Kanpur
Patna
Varanasi
Bhopal
India
Indore
Nagpur
Bangladesh
Daulatpur
Kolkata
Surat
Mumbai
Hyderabad
Myanmar
(Burma)
Guiyang
贵阳市
Changsha
长沙市
Kunming
昆明市
Ha Noi
Laos
Haikou
海口市
Taiwan
Hong Kong
香港
澳門
North
Korea
South
Korea
Fukuoka
福岡
Osaka
大阪
Hiroshima
広島
Kumamoto
熊本
Miyazaki
宮崎
Naha
那覇
RST
s0
s1
s10
s50
s100

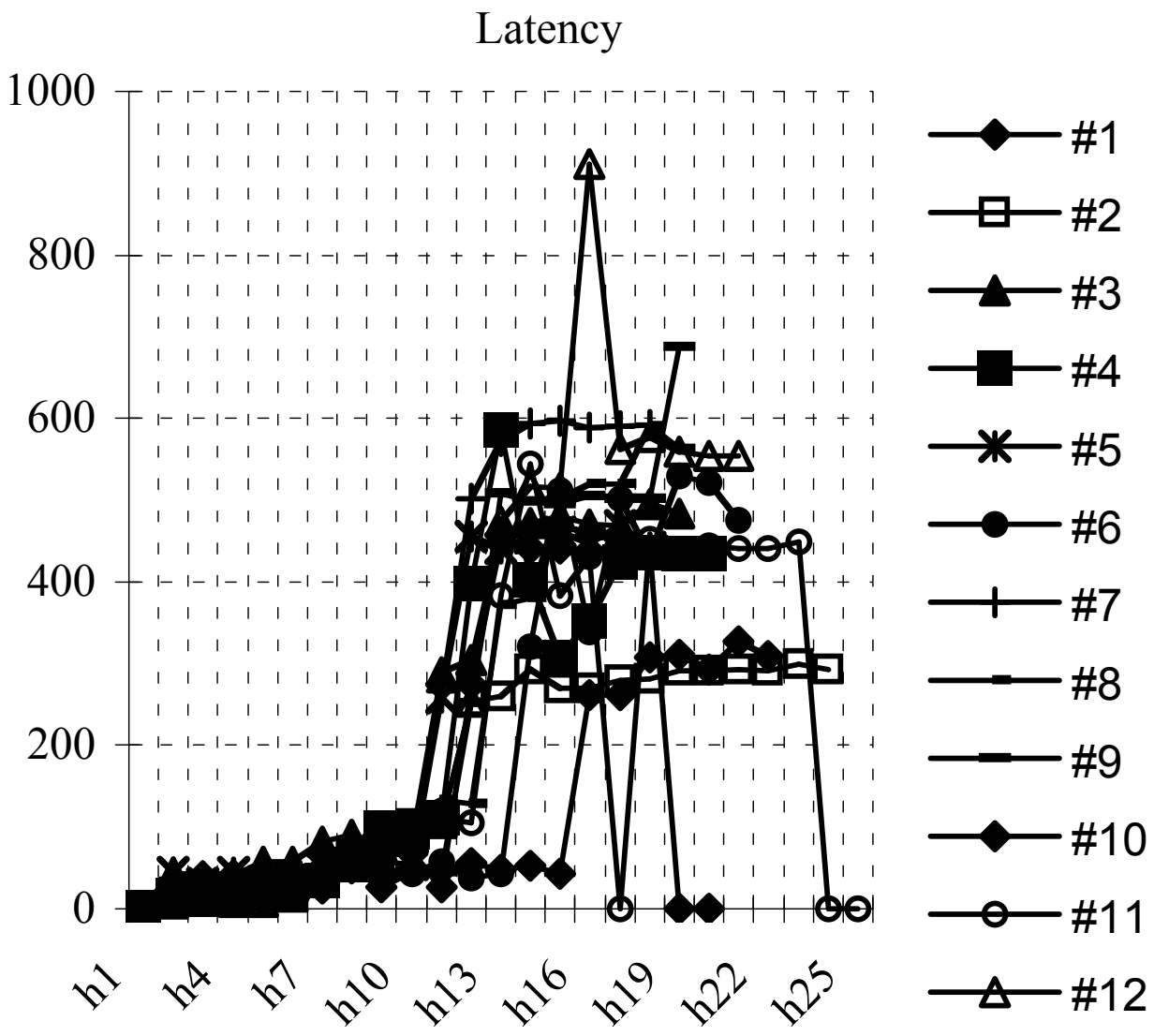
STD DEV


STDDEV-AVG of
#keywords
350
300
250
200
150
100
50
0
#1
#2
#4
#5
#6
#7
#8
#9
#10
#11

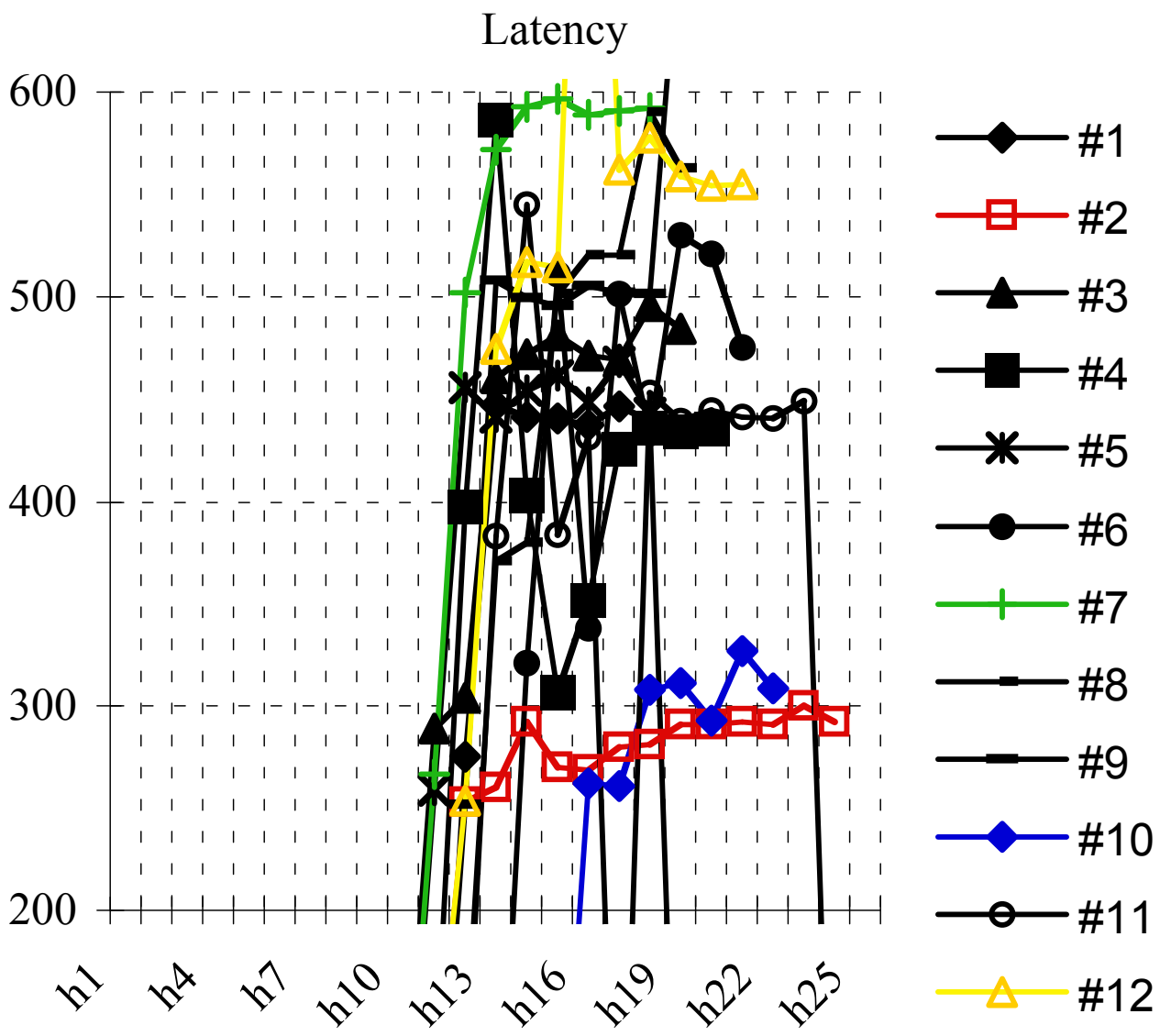
STD DEV


Zoom-up of STDDEV-AVG
of #keywords by location
50
40
30
20
10
0
#1
#2
#4
#5
#6
#7
#8
#9
#10
#11

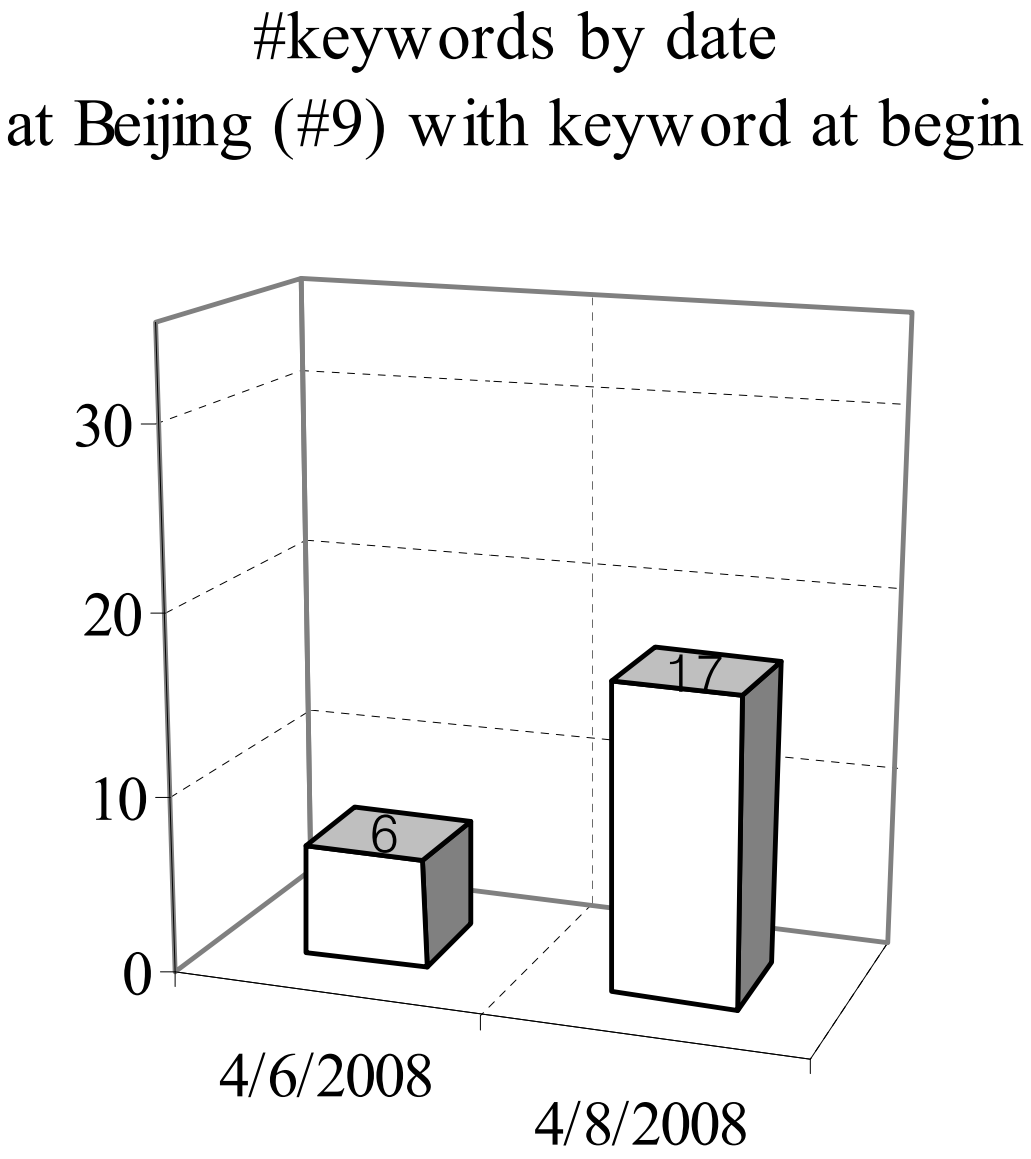
Latency


Latency
1000
800
600
400
200
0
h1
h4
h7
h10
h13
h16
h19
h22
h25
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12

Latency


Latency
600
500
400
300
200
h1
h4
h7
h10
h13
h16
h19
h22
h25
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12

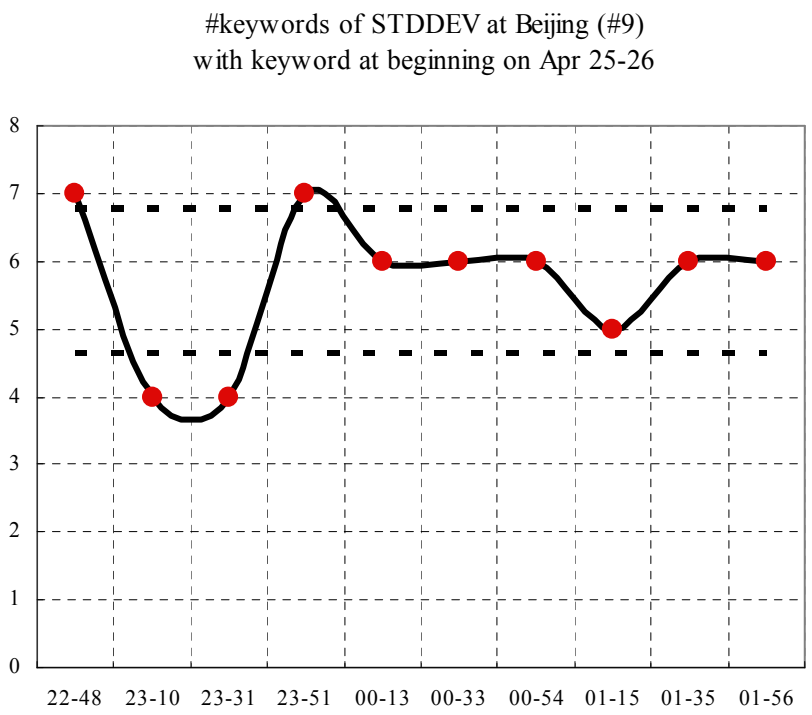
Beijing on 4KB



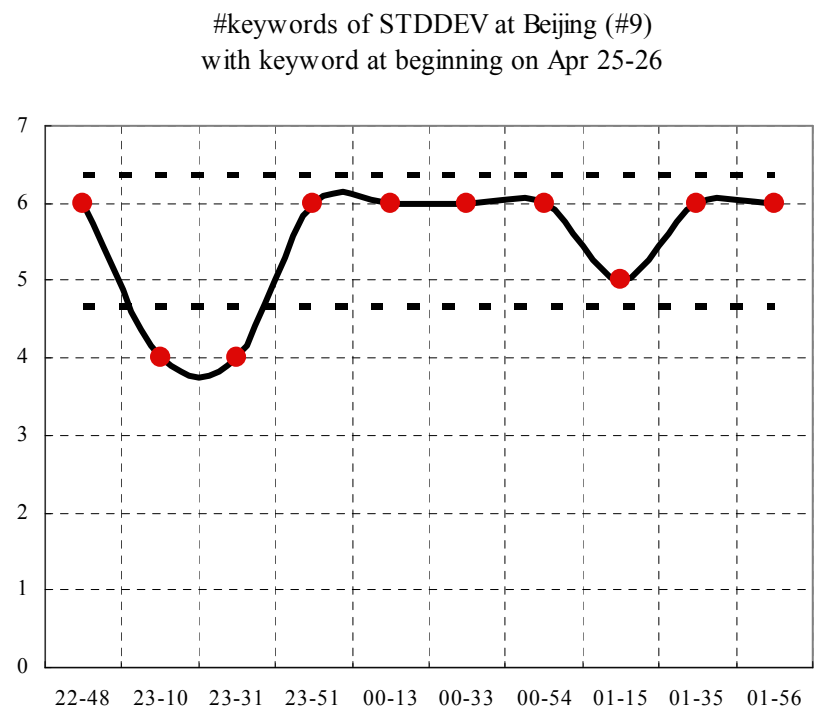
False Positive

- Hard to predict, huh?
  - Need more time/tests
- Assume all odd packets false positive
  - Let’s visit all graphs again!
- False positive rate of ~8.5%
  - 2236 times not rec’d in total since Mar
  - 2045 RSTs observed

STD DEV

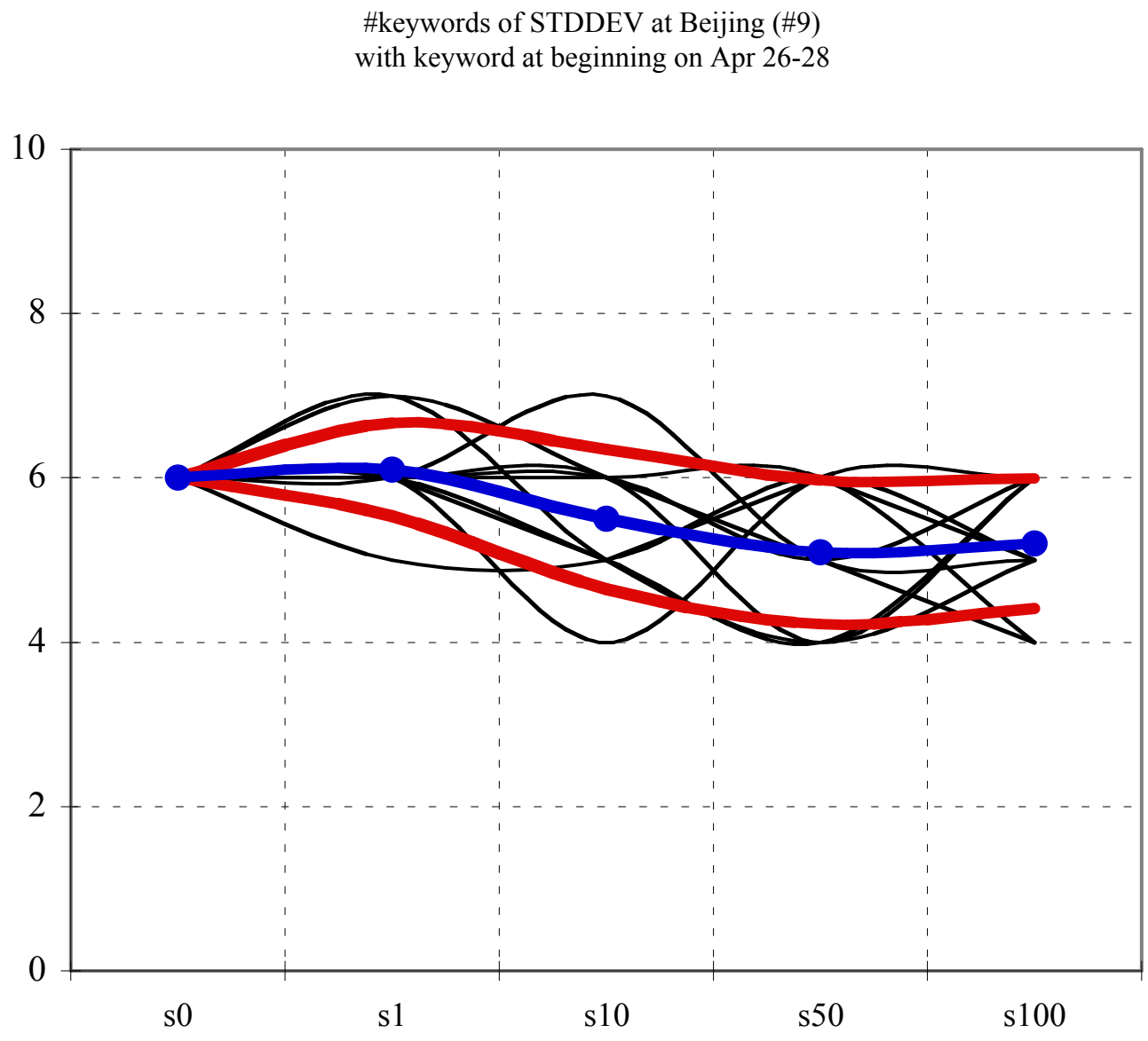


RST+Others

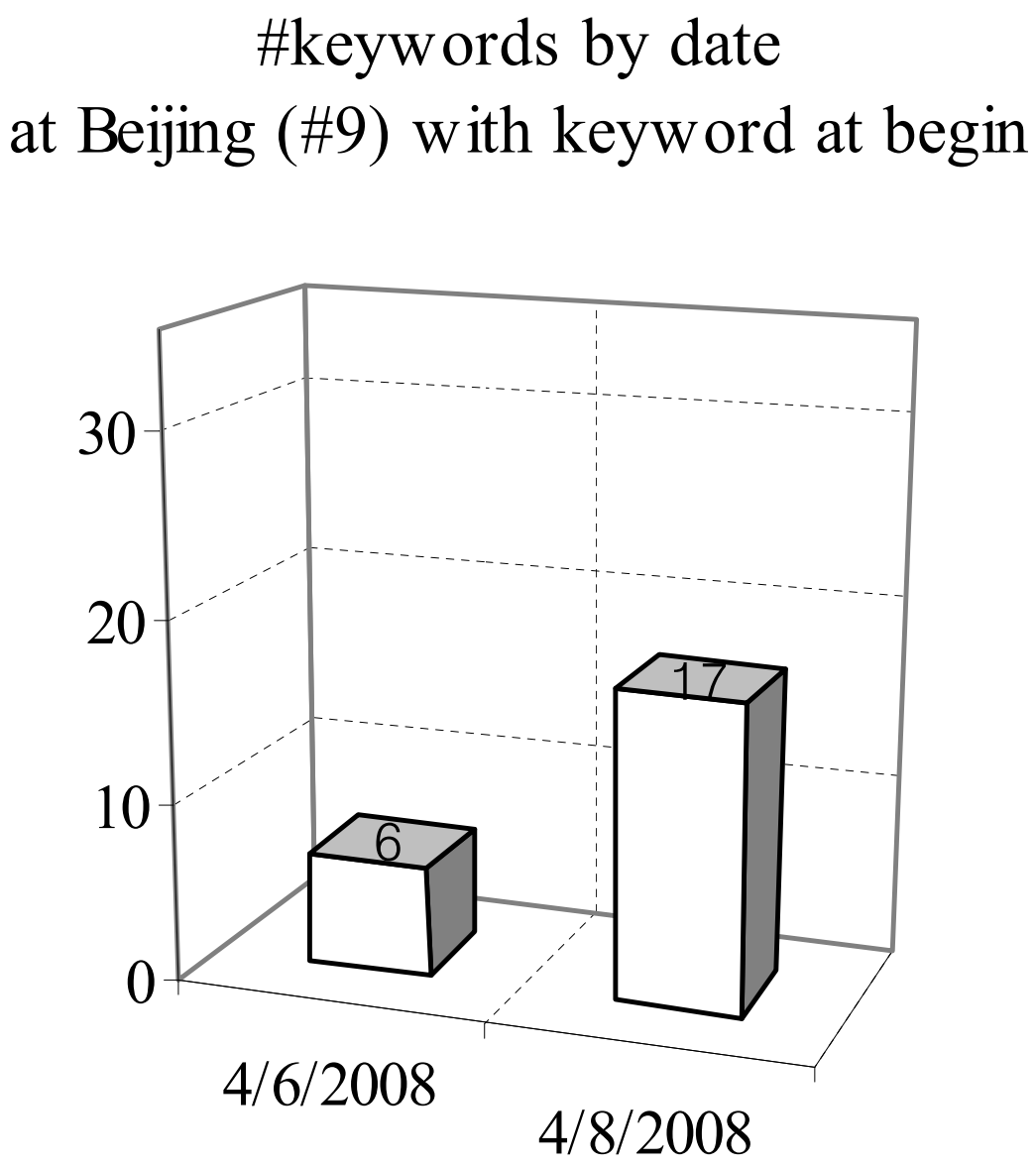


RST only

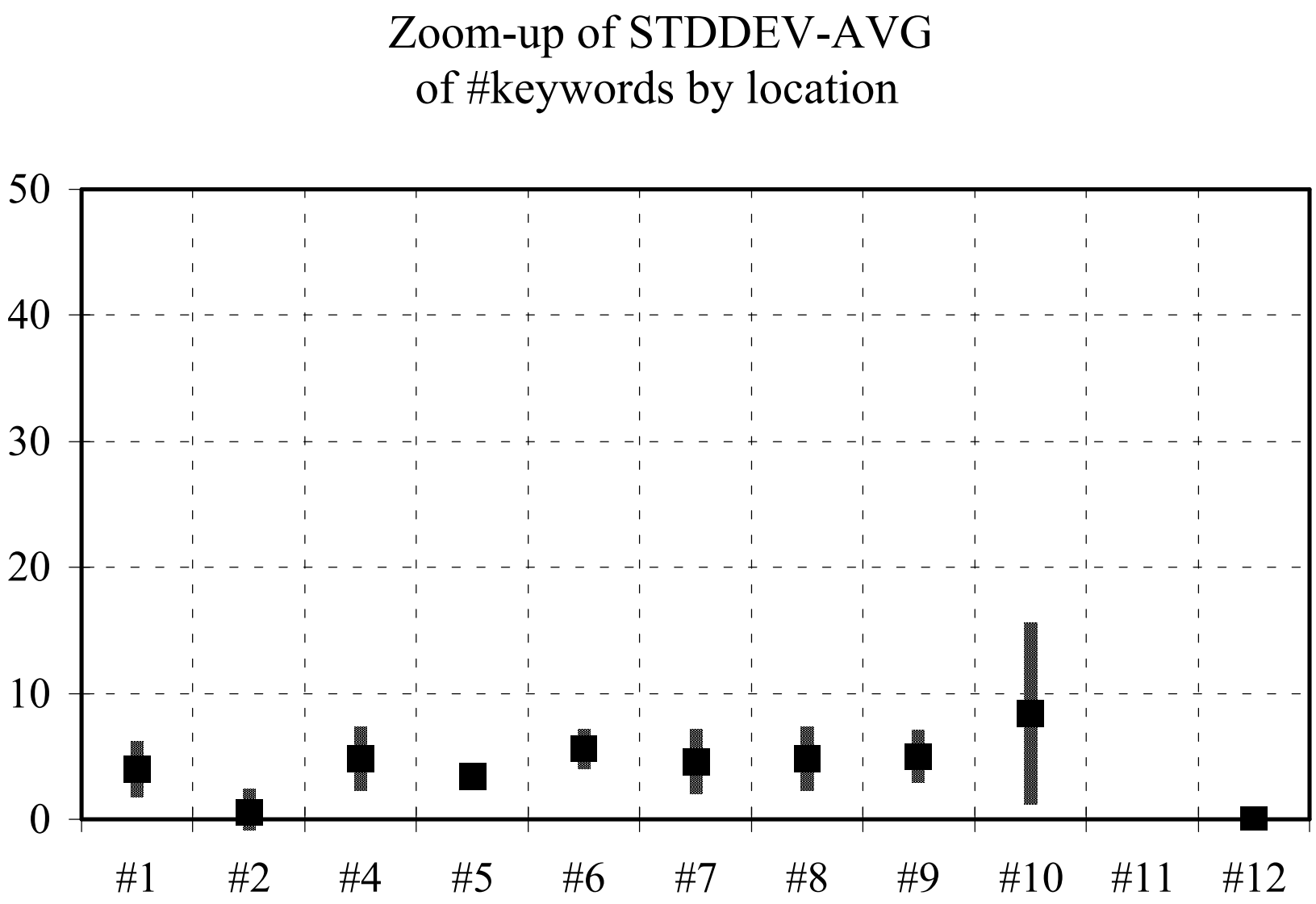
STD DEV


#keywords of STDDEV at Beijing (#9)
with keyword at beginning on Apr 26-28
10
8
6
4
2
0
s0
s1
s10
s50
s100

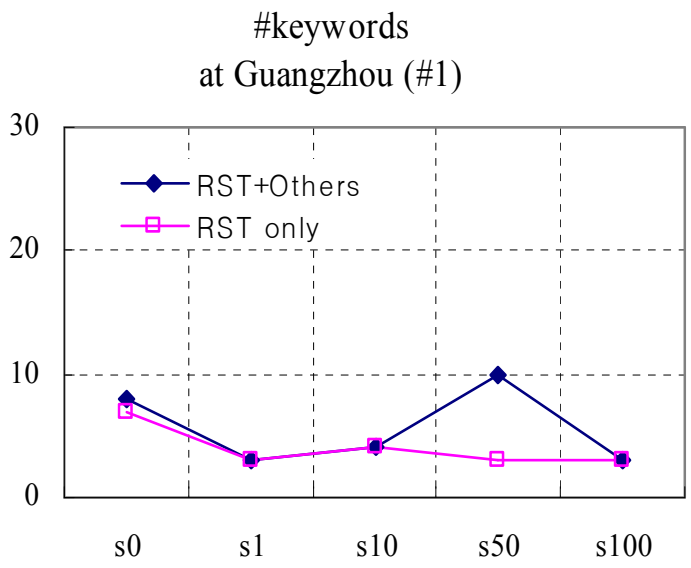
Beijing on 4KB

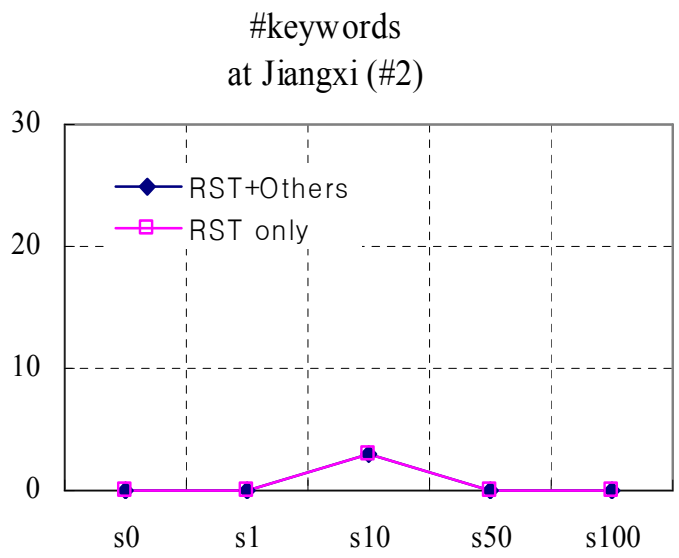


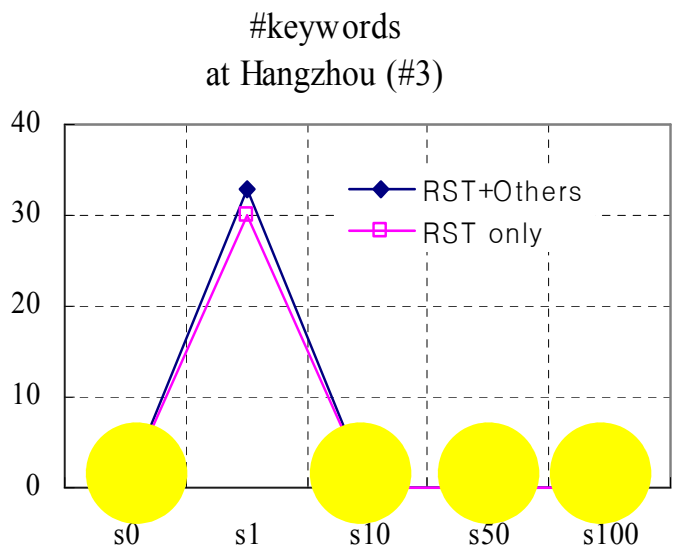
STD DEV

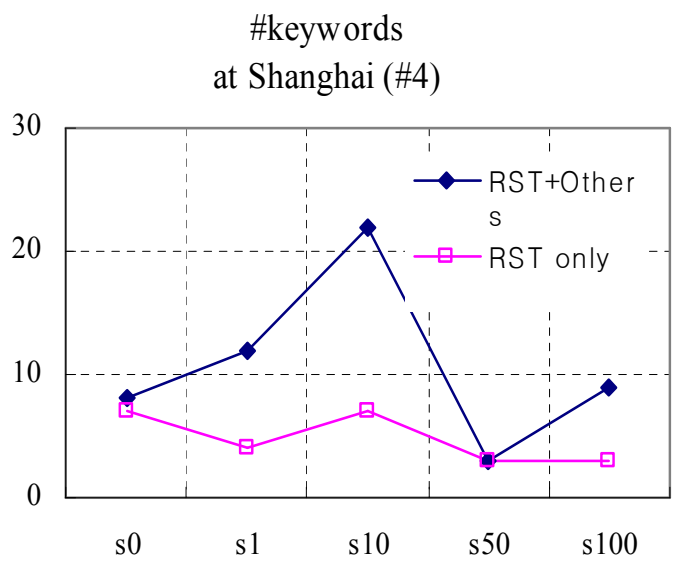

Zoom-up of STDDEV-AVG
of #keywords by location
50
40
30
20
10
0
#1
#2
#4
#5
#6
#7
#8
#9
#10
#11
#12

By Location

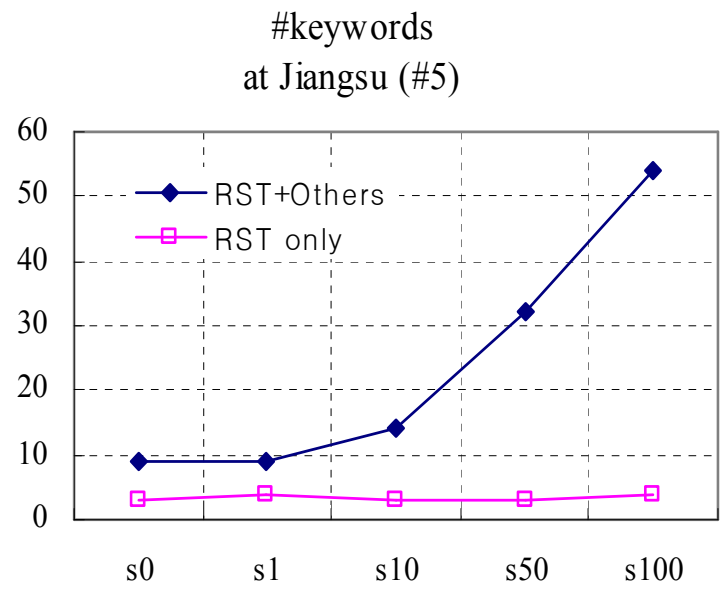

#keywords
at Guangzhou (#1)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

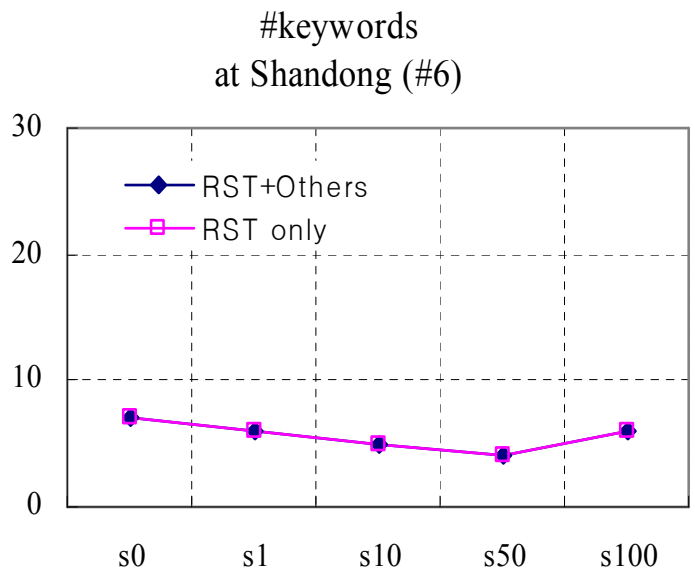

#keywords
at Jiangxi (#2)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

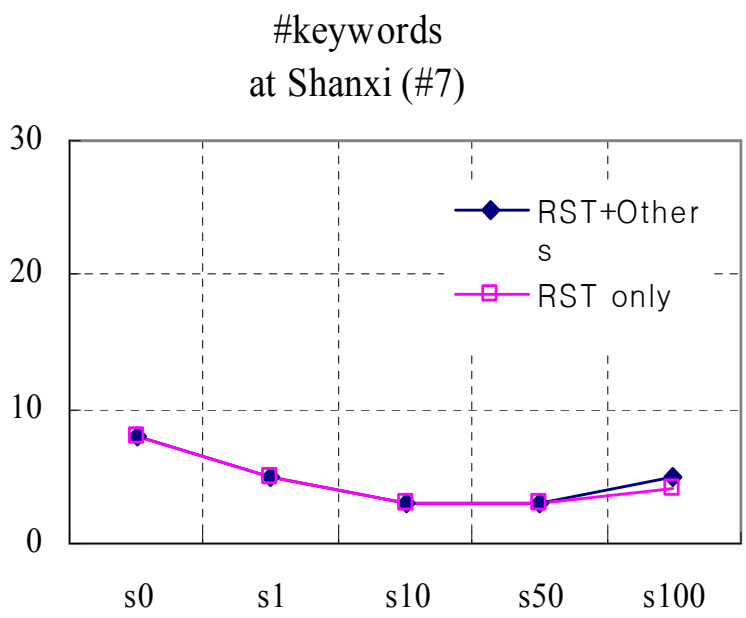

#keywords
at Hangzhou (#3)
40
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

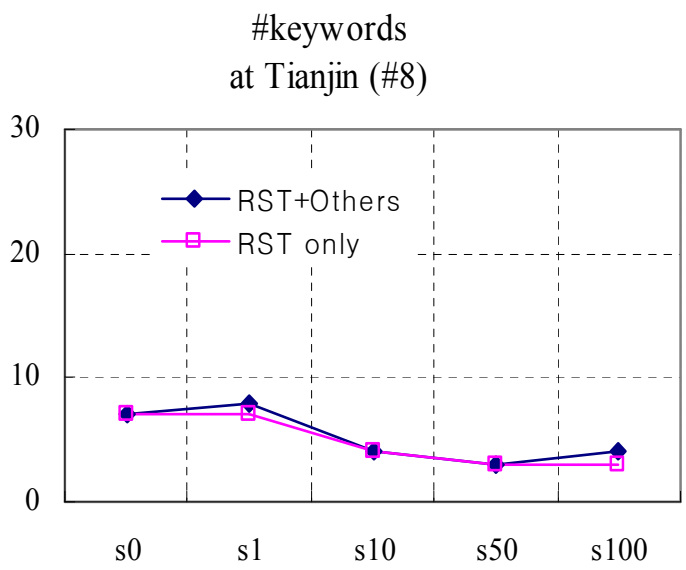

#keywords
at Shanghai (#4)
30
20
10
0
RST+Other
s
RST only
s0
s1
s10
s50
s100

By Location

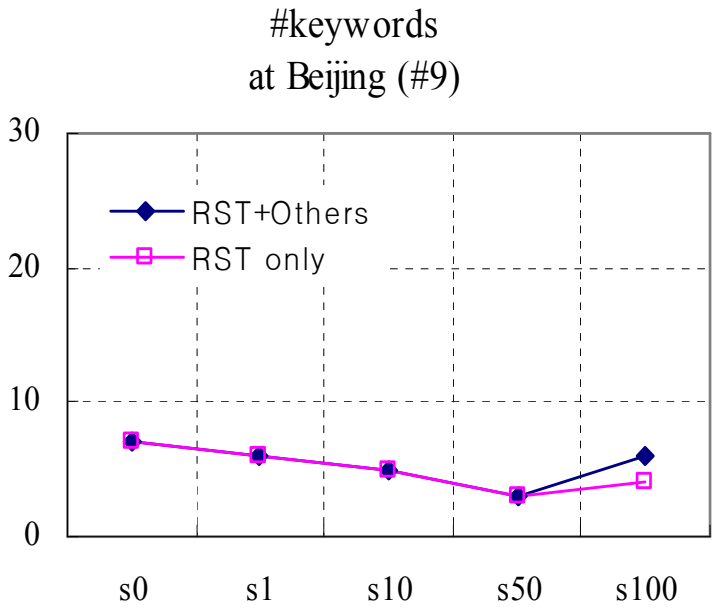

#keywords
at Jiangsu (#5)
60
50
40
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

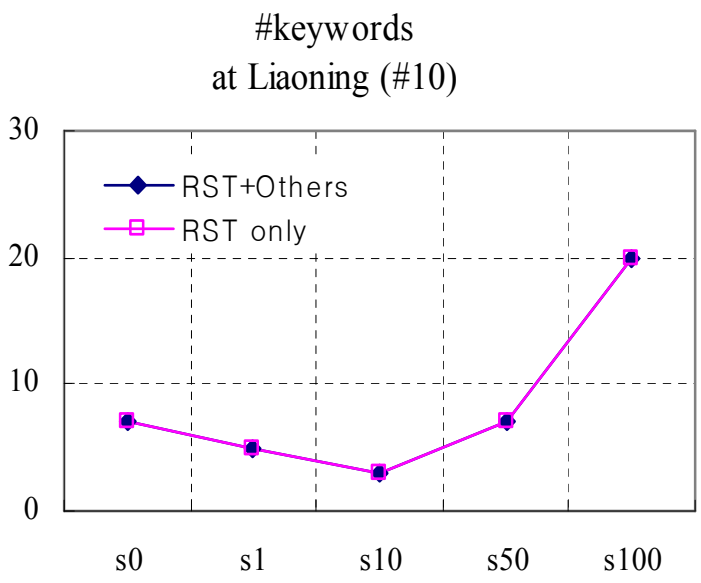

#keywords
at Shandong (#6)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

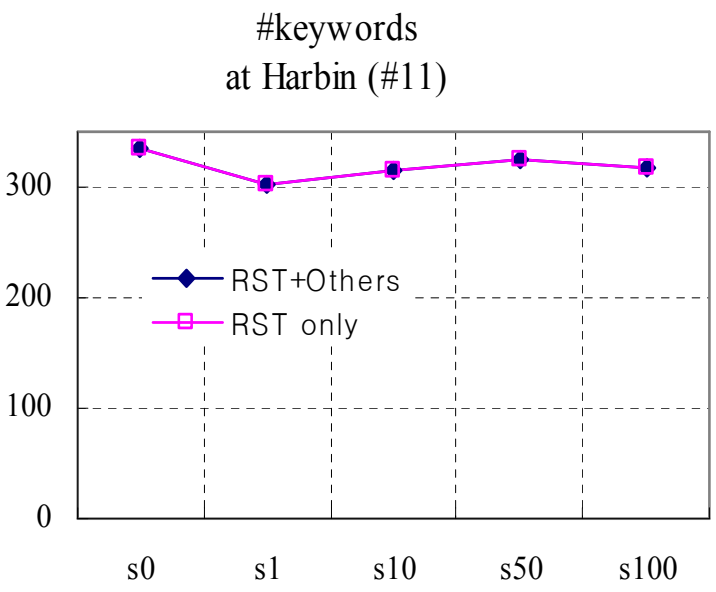

#keywords
at Shanxi (#7)
30
20
10
0
RST+Other
s
RST only
s0
s1
s10
s50
s100

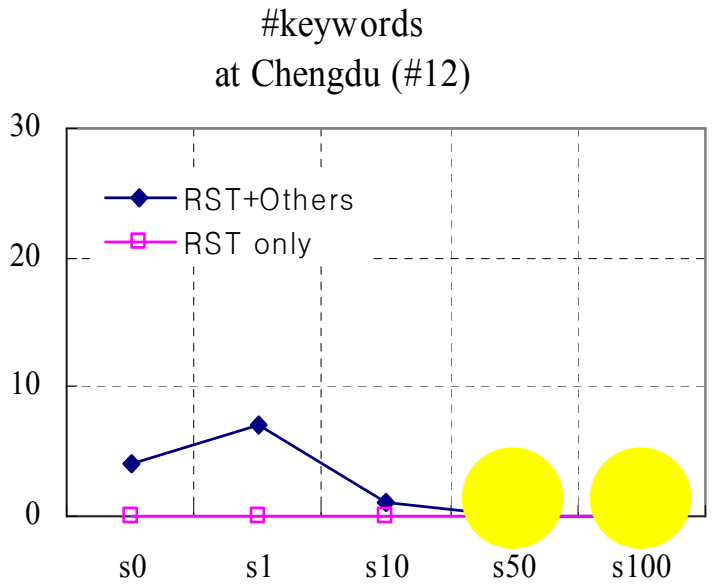

#keywords
at Tianjin (#8)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

By Location

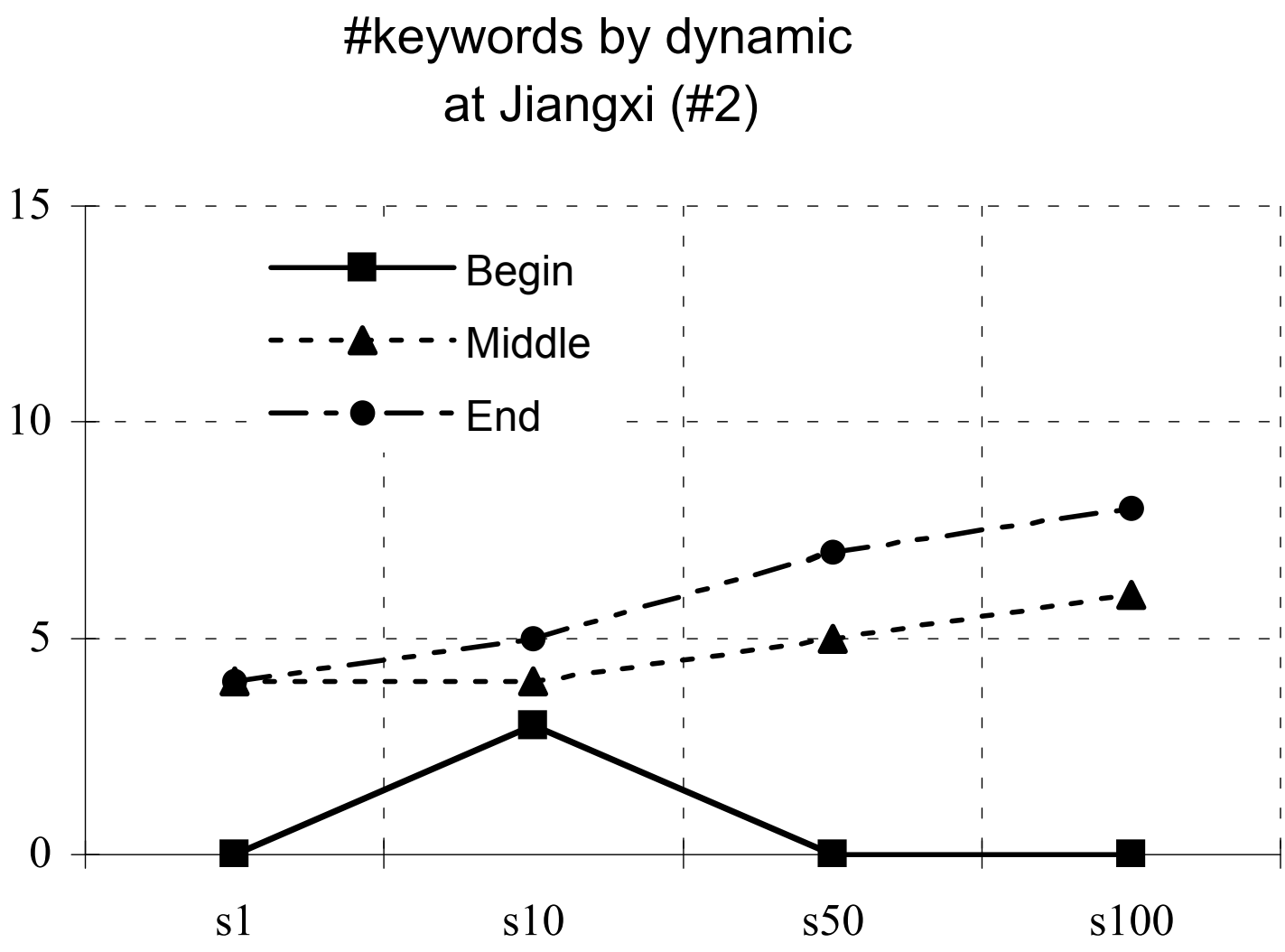

#keywords
at Beijing (#9)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100


#keywords
at Liaoning (#10)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

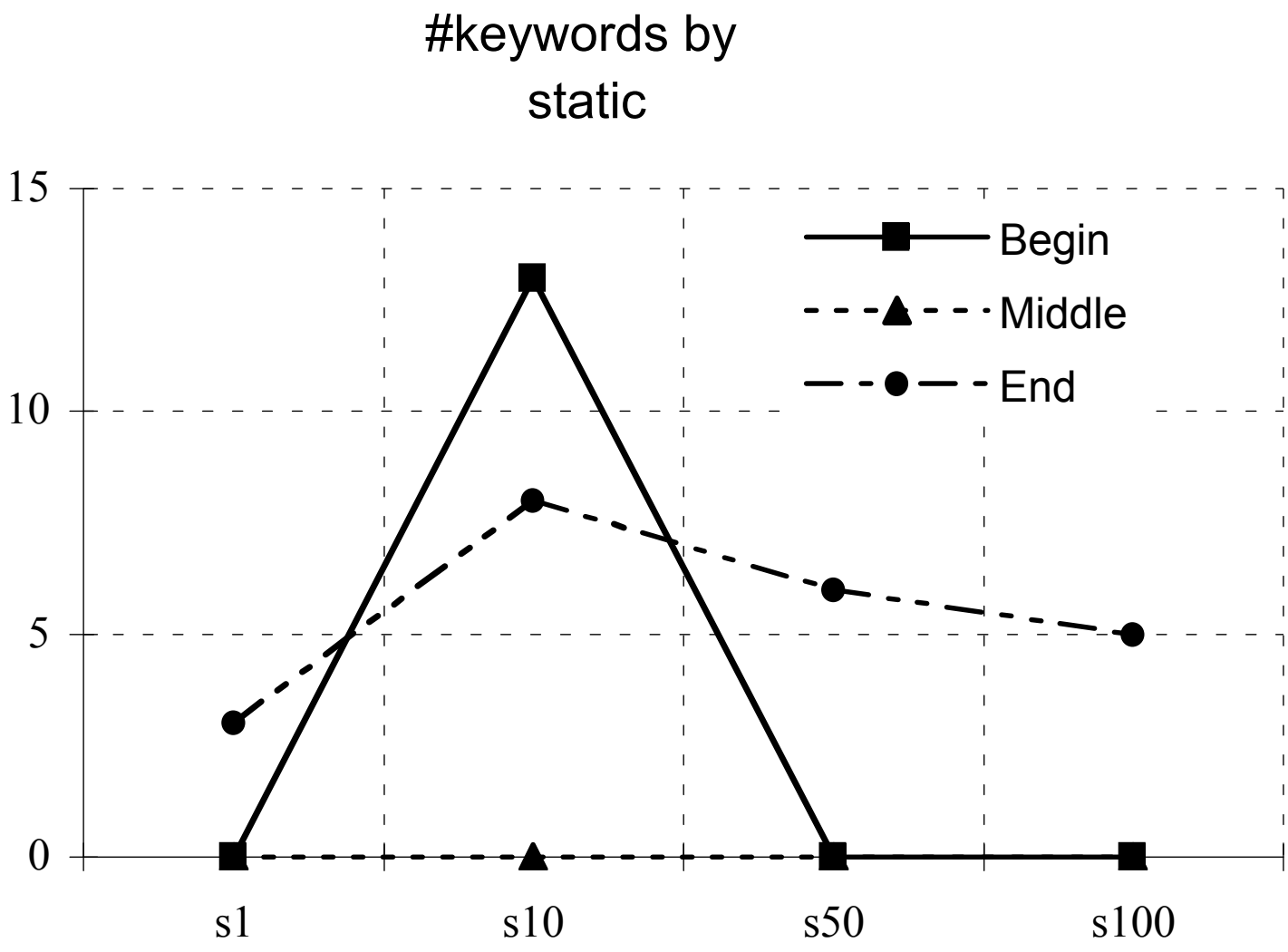

#keywords
at Harbin (#11)
300
200
100
0
RST+Others
RST only
s0
s1
s10
s50
s100


#keywords
at Chengdu (#12)
30
20
10
0
RST+Others
RST only
s0
s1
s10
s50
s100

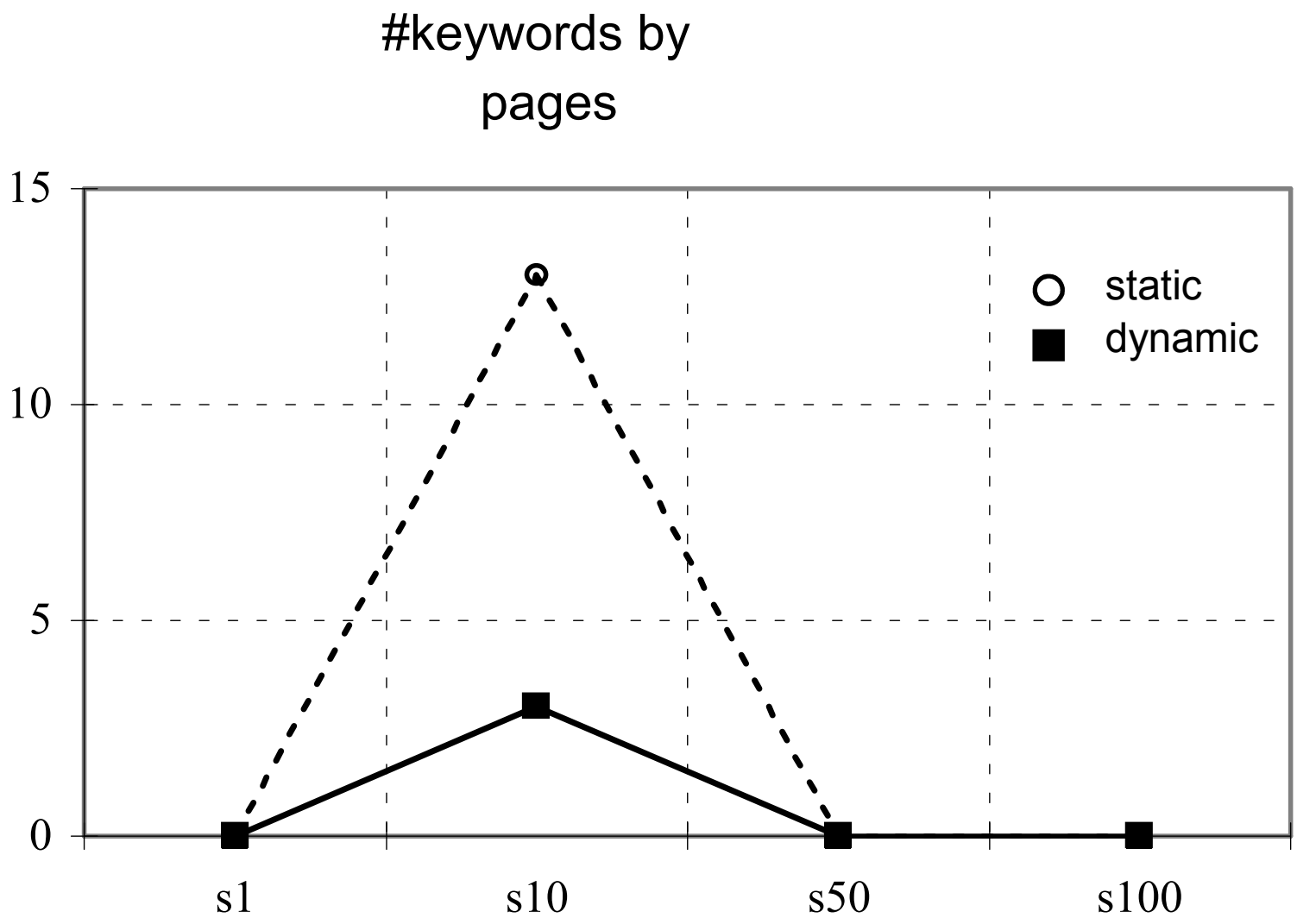
Dynamic Page (Beijing)



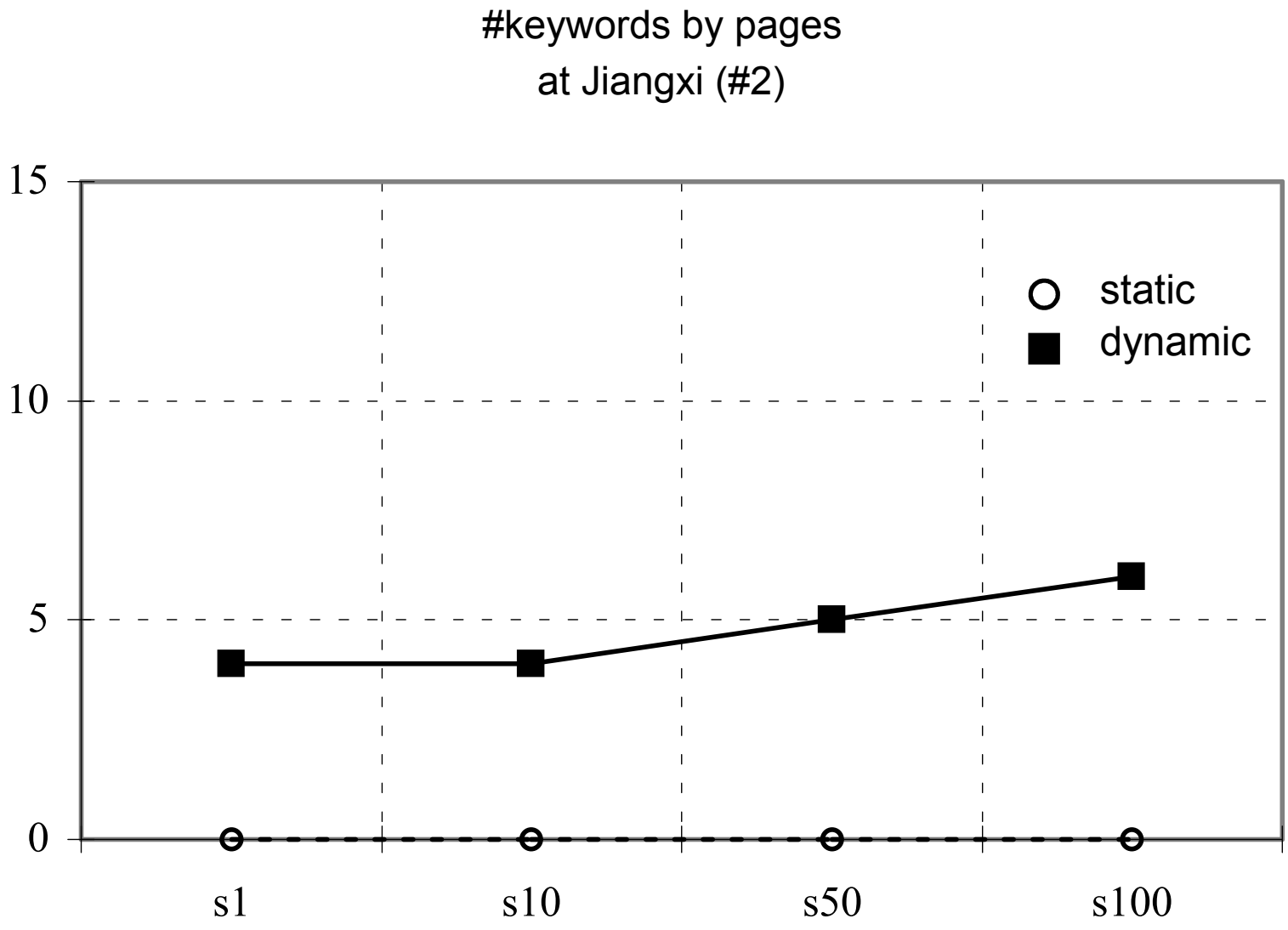
Static Page (Beijing)



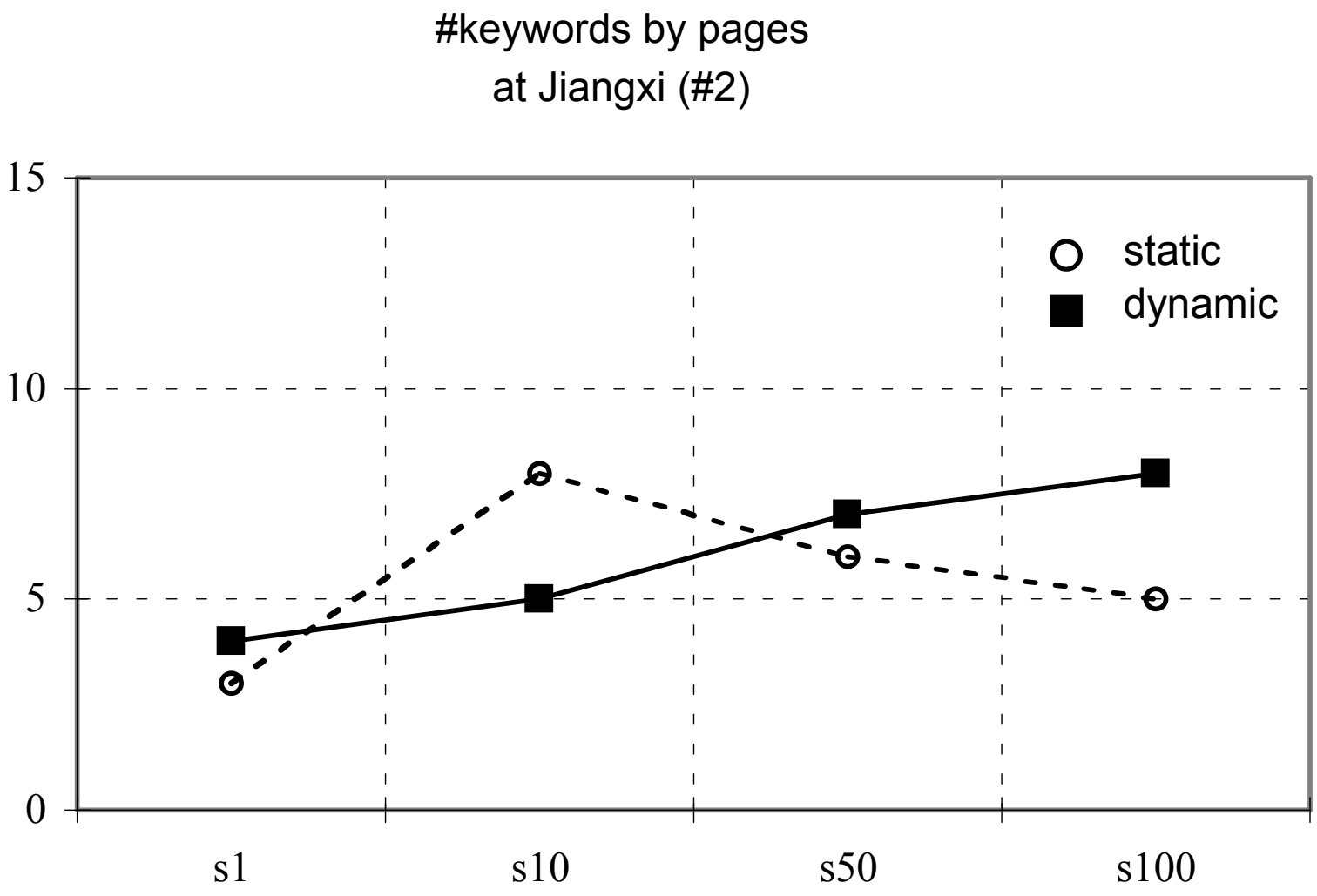
Dynamic vs Static: beginning



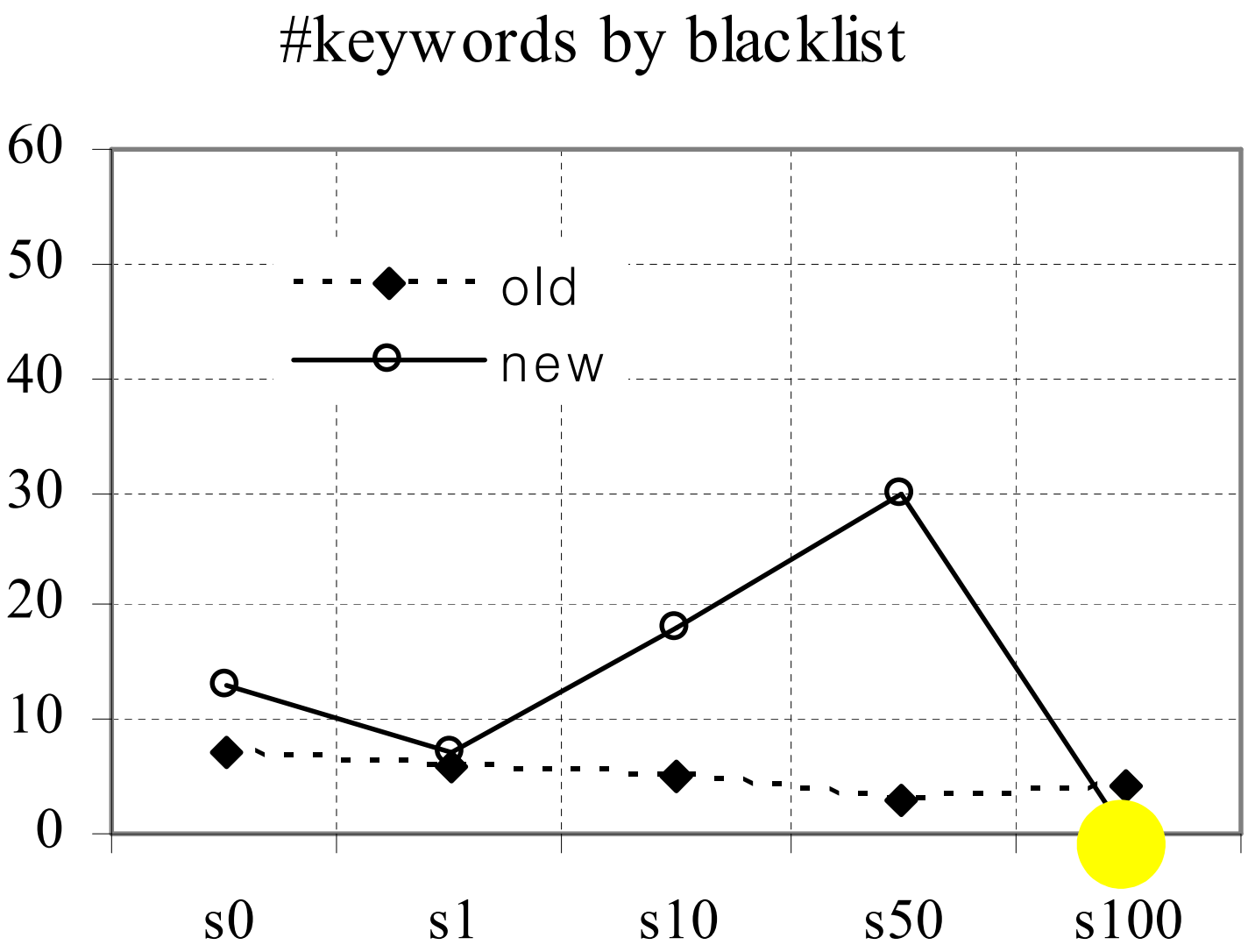
Dynamic vs Static: middle



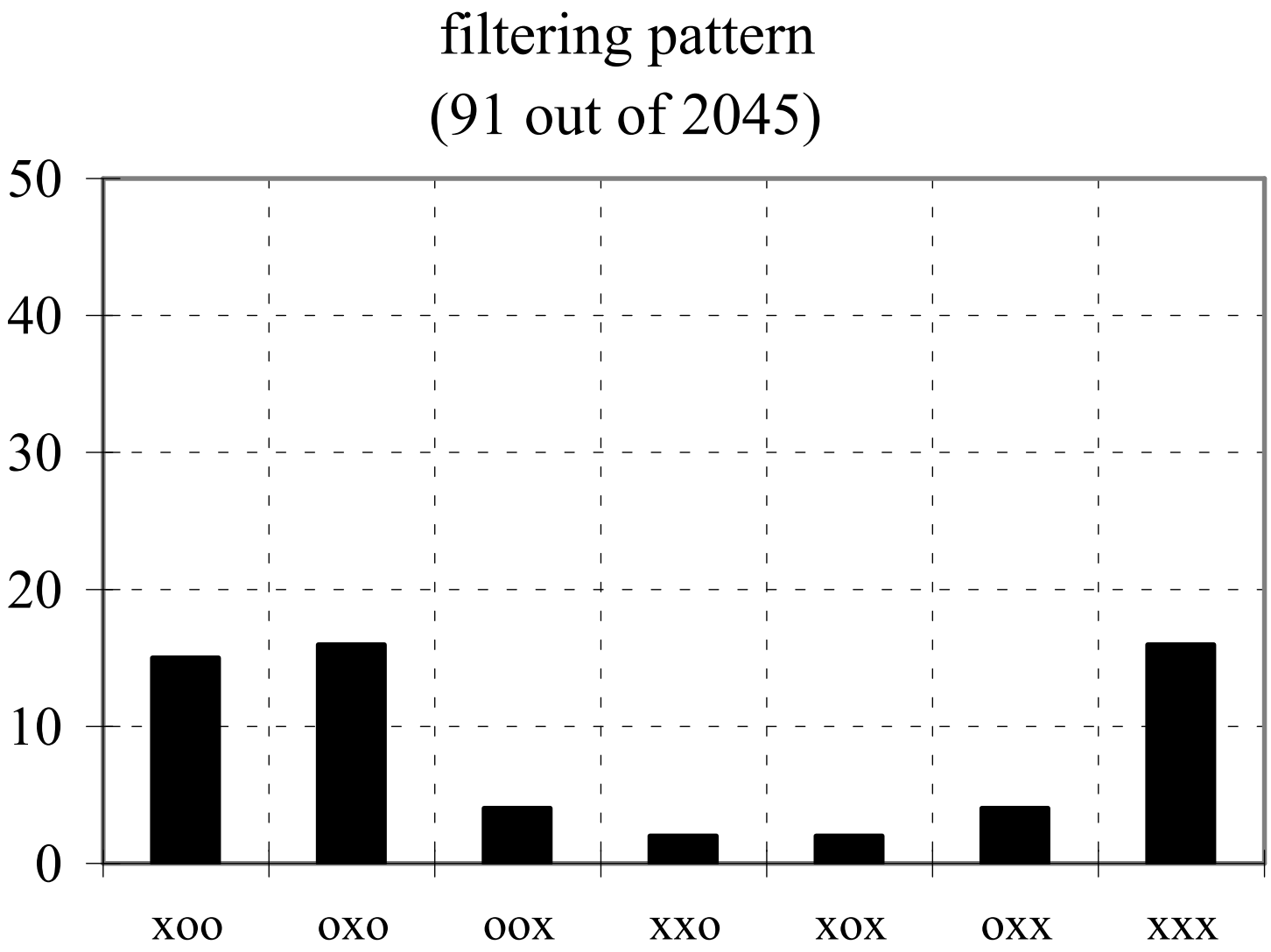
Dynamic vs Static: end



Blacklist updated



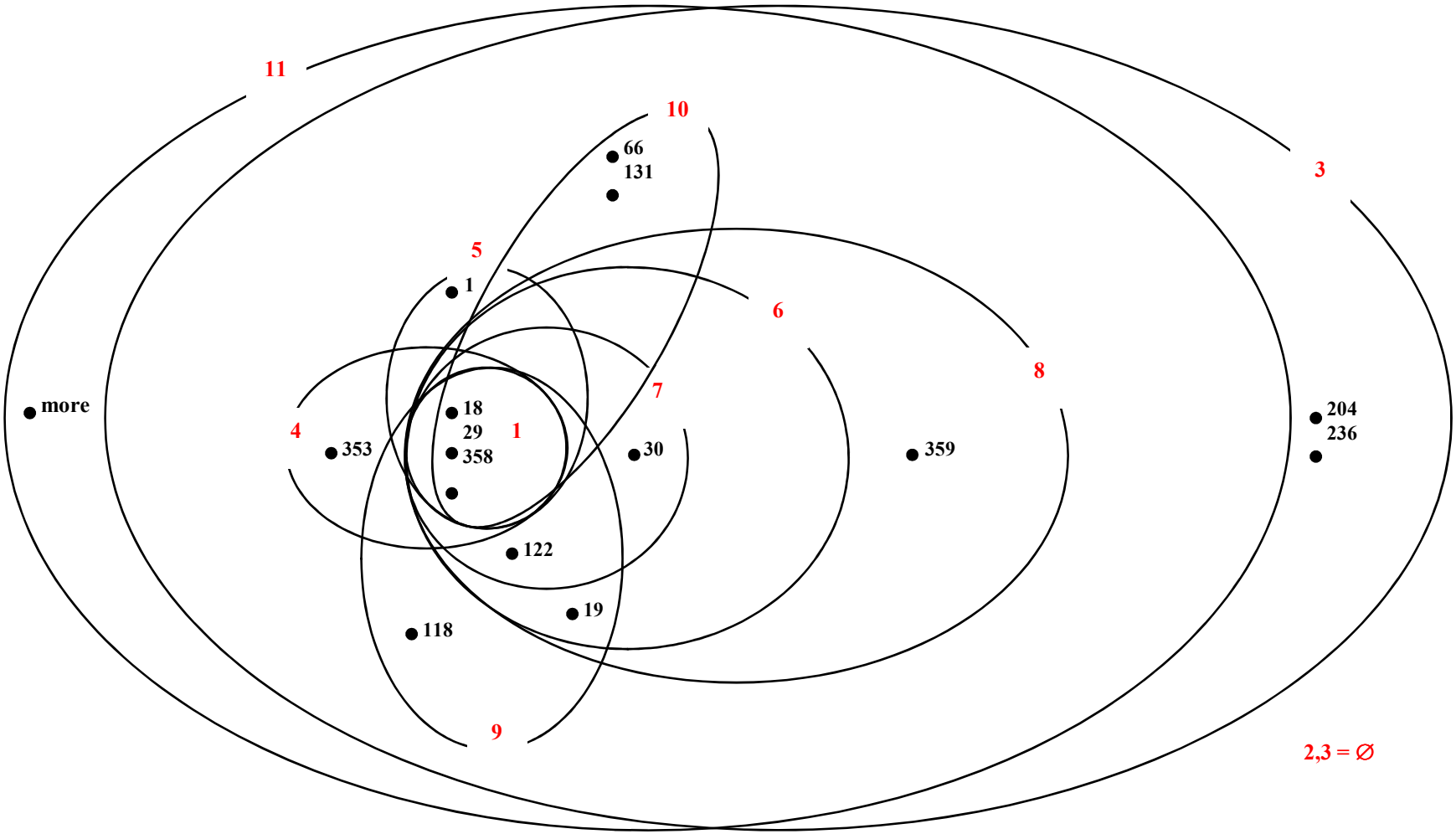
Filtering Pattern: probabilistic?



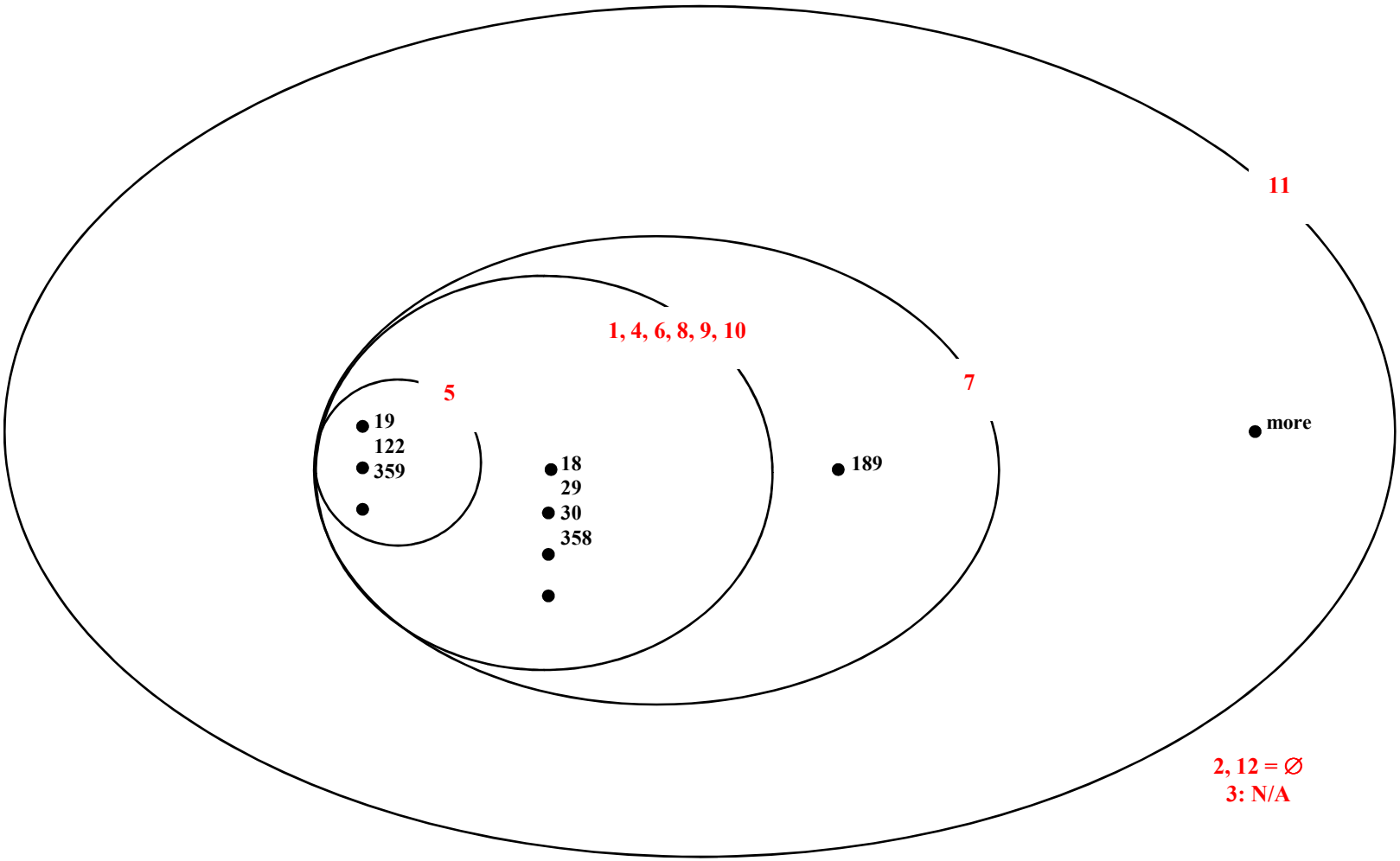
Blacklist

- ~95% blacklist keywords of HTML GET request also blocked in HTML response
  - ~54% blocked when excluding Harbin
- Less substantial blocking on HTML response
  - Probably due to overhead and/or latency
- Unlike request, response blacklist seems to be distributed over the nation
  - Some GFCs seem to share

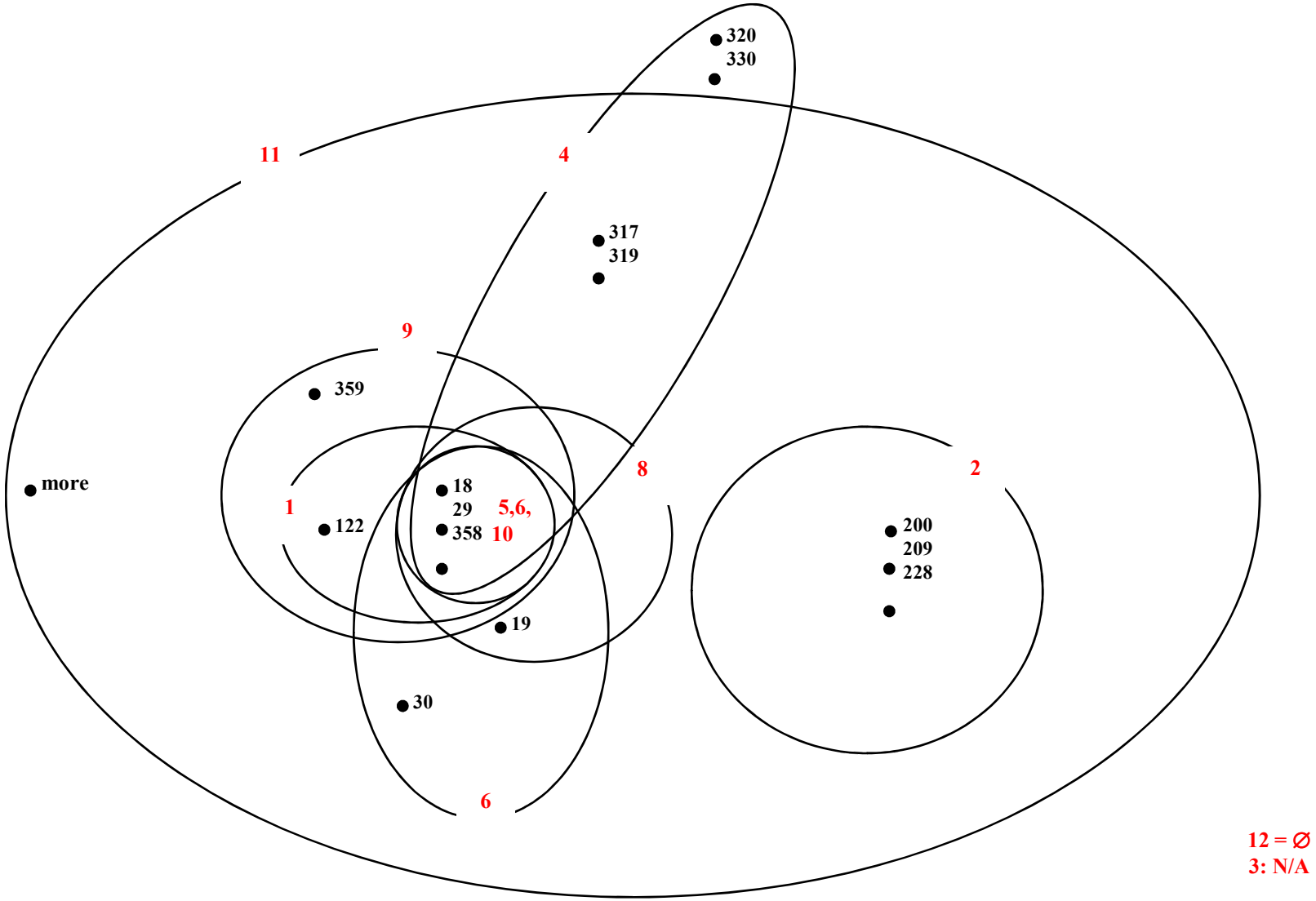
Blacklist (4KB)



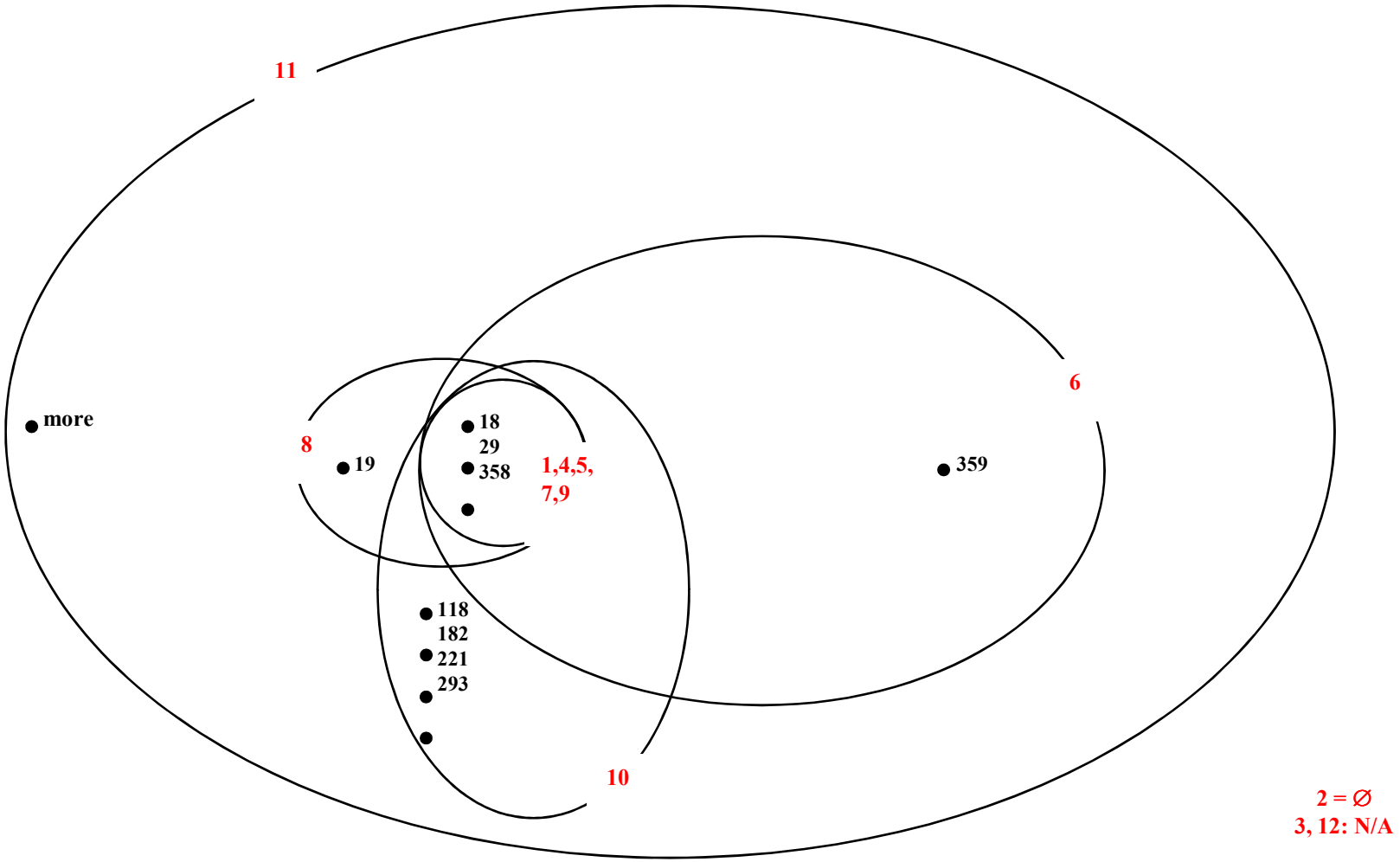
Blacklist (600B)



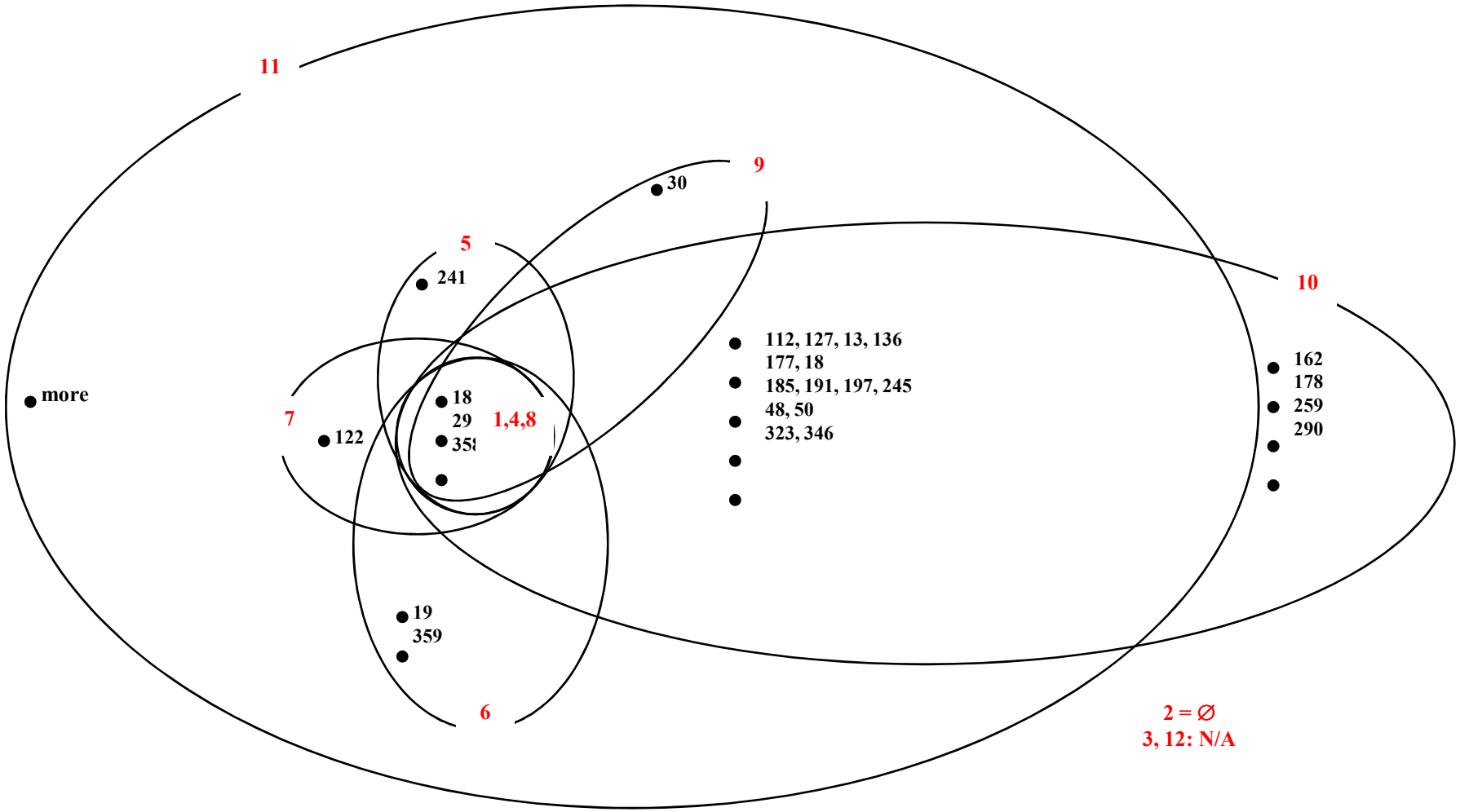
Blacklist (40KB)



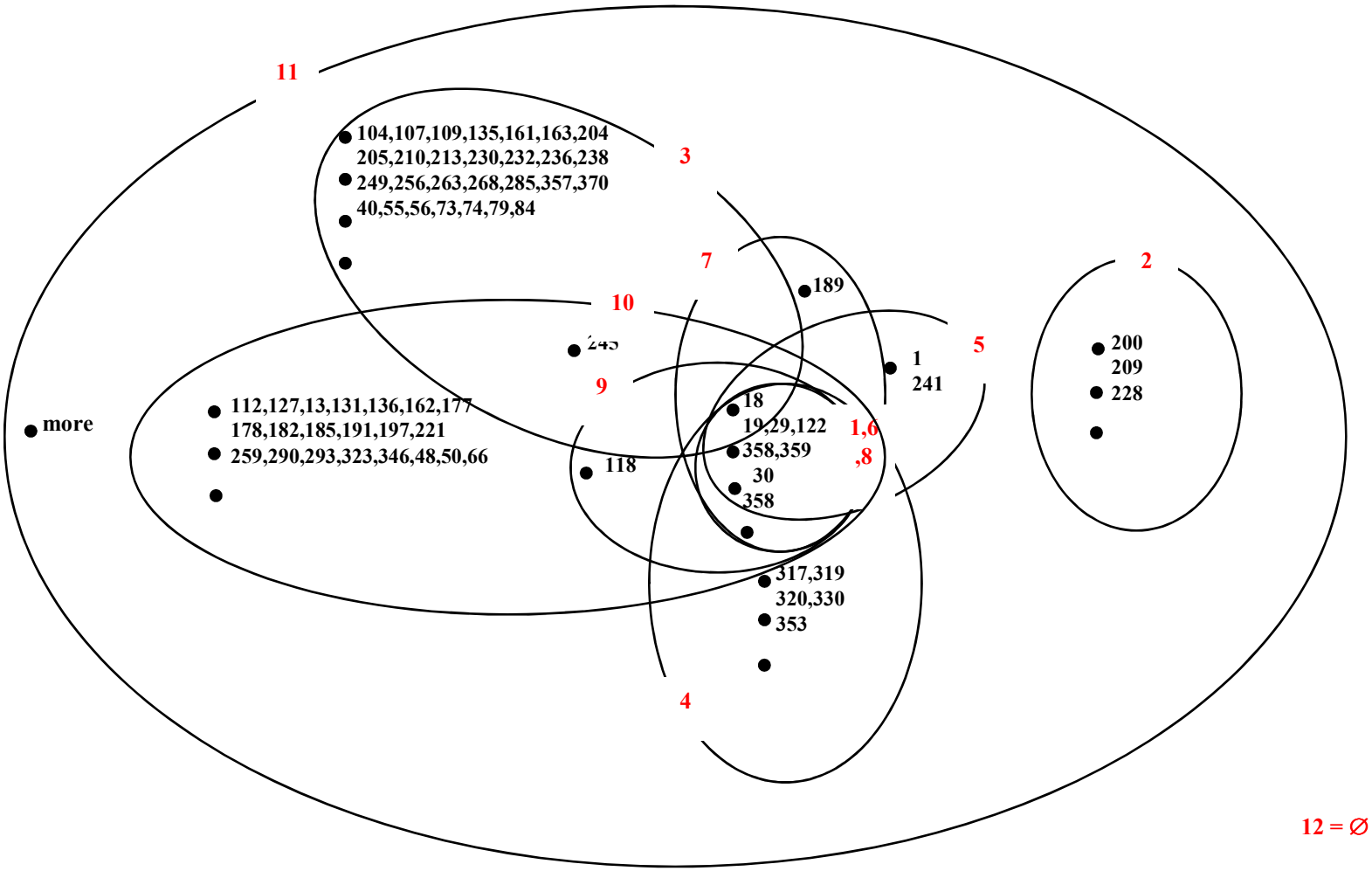
Blacklist (175KB)



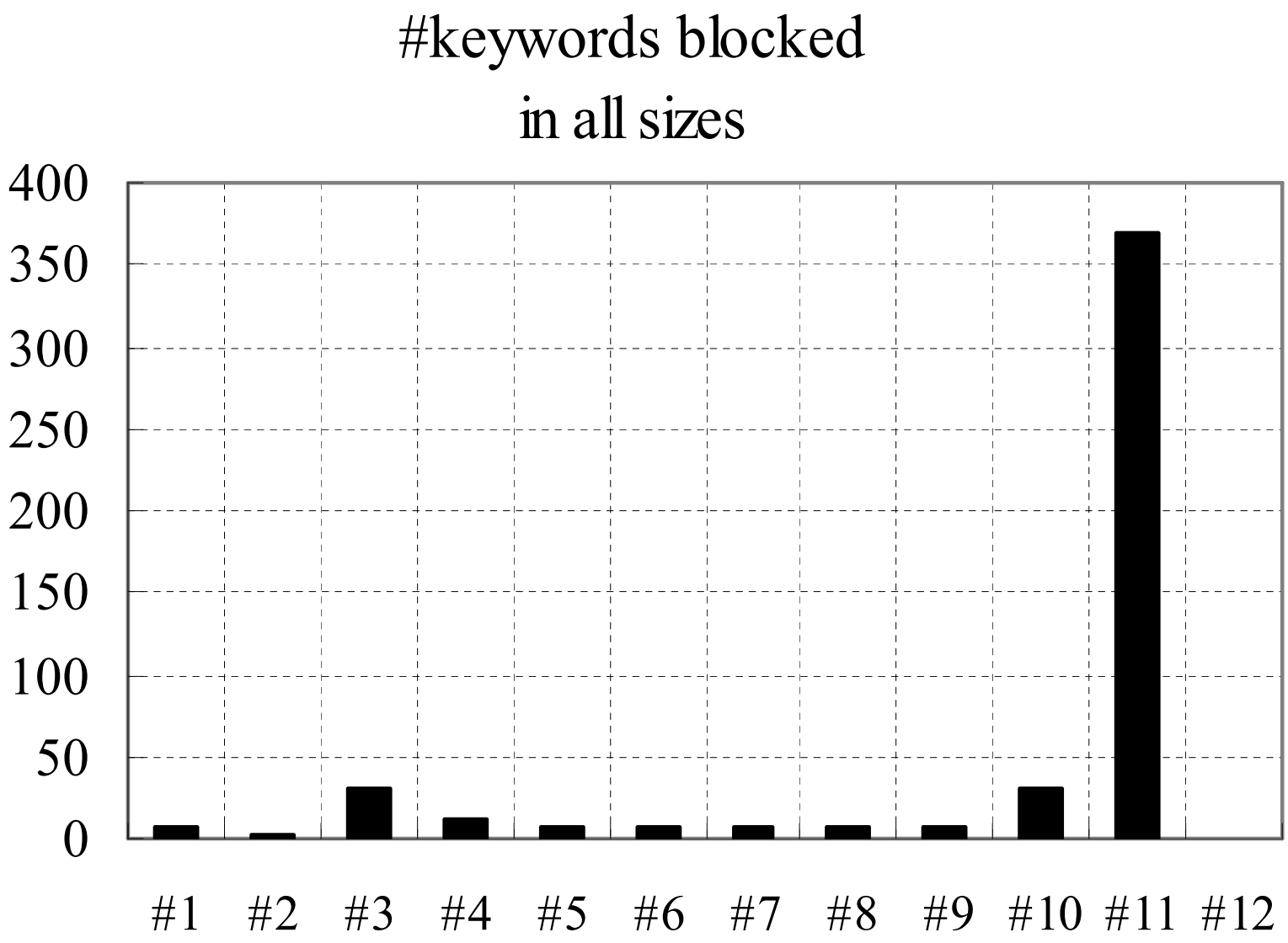
Blacklist (325KB)



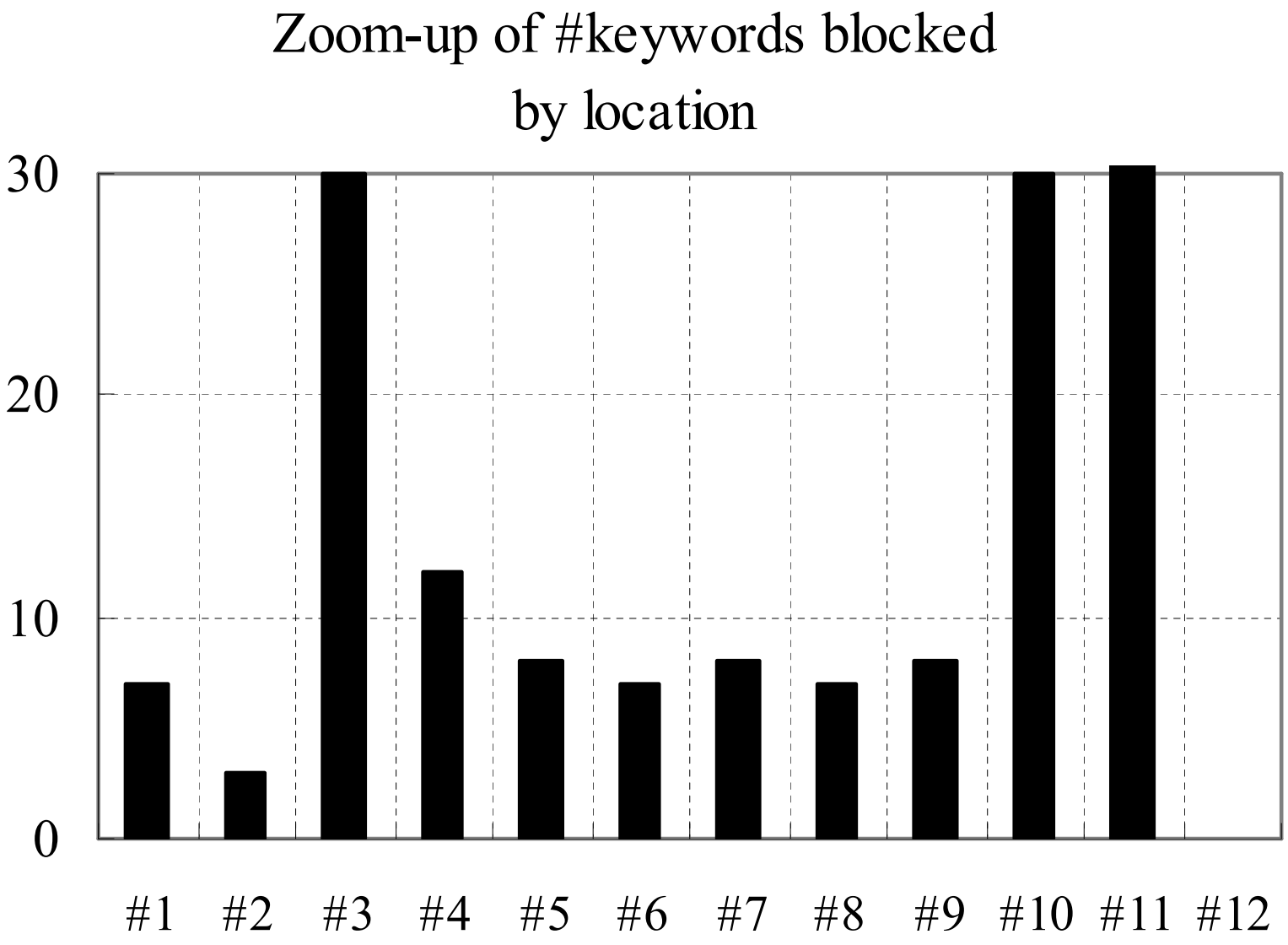
Blacklist (all)



Blacklist Size



Blacklist Size



Conclusion

- GFC performs symmetric filtering against HTML
  - Less substantial on HTML response probably due to overhead/latency
  - Distributed blacklist words
  - Share a subset of blacklist words
  - Route path, packet size, and keyword location affect censorship efficiency
  - Stateful vs. stateless TCP RSTs
  - Inversion suggests application-layer censorship???

Thank you

- Any ***questions*** or ***feedbacks***?

Lab 4

- We’re not using an actual IDS, but by iptables.
  - the server is too simple and is just buffering up to 4 KB
  - can beat this by sending a sequence of packets which split keywords
    - need to flush first before sending the next
    - otherwise, TCP RSTs will be triggered
- Lab 4 1/2 will be about TCP fragmentation(?)

Socket in Python

- import socket # client
- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- s.connect((“10.0.0.3”, 8080))
- s.send(“hello”)
- d=s.recv(1024)
- print repr(d)
- s.close()

Scapy

- TCP fragmentation?
  - Not a proper jargon! But, TCP supports byte streams and out-of-order delivery.
    - Let’s call it for the sake of simplicity.
  - Why not IP fragmentation? RFC1858!
    - Tiny fragment attack
    - can use fragroute for smaller IP packets
- can forge packets with Scapy

Scapy (cont'd)

- Use whatever you like, for example MuxTCP, C code with libpcap, etc. (Visit **Milw0rm** for code.)
- Scapy is troublesome in initiating TCP
  - need to change firewall rules by iptables
  - iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
  - wouldn't work for Lab 4 1/2 since the server is not under control
    - initiate TCP handshake with your own socket

Scapy (cont'd)

- To use tcpdump, you need to set PATH
  - use scapy.sh script mailed
  - use sniff.py to sniff the network
- Need to know some TCP/IP stuff
  - IP
    - IPID, IP addresses
  - TCP
    - flags, sequence #, ack #, ports

TCP Handshake

- fd = open(“message.txt”, ‘r’)
- lines = fd.readlines()
- fd.close()

- for l in lines:
- print l.strip()+’\n’

- import time
- time.sleep(1)