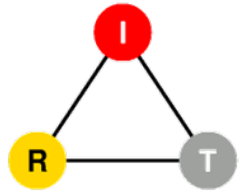
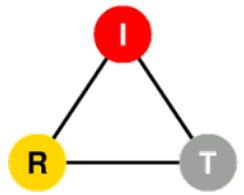
SipCloud: Dynamically Scalable SIP Proxies in the Cloud

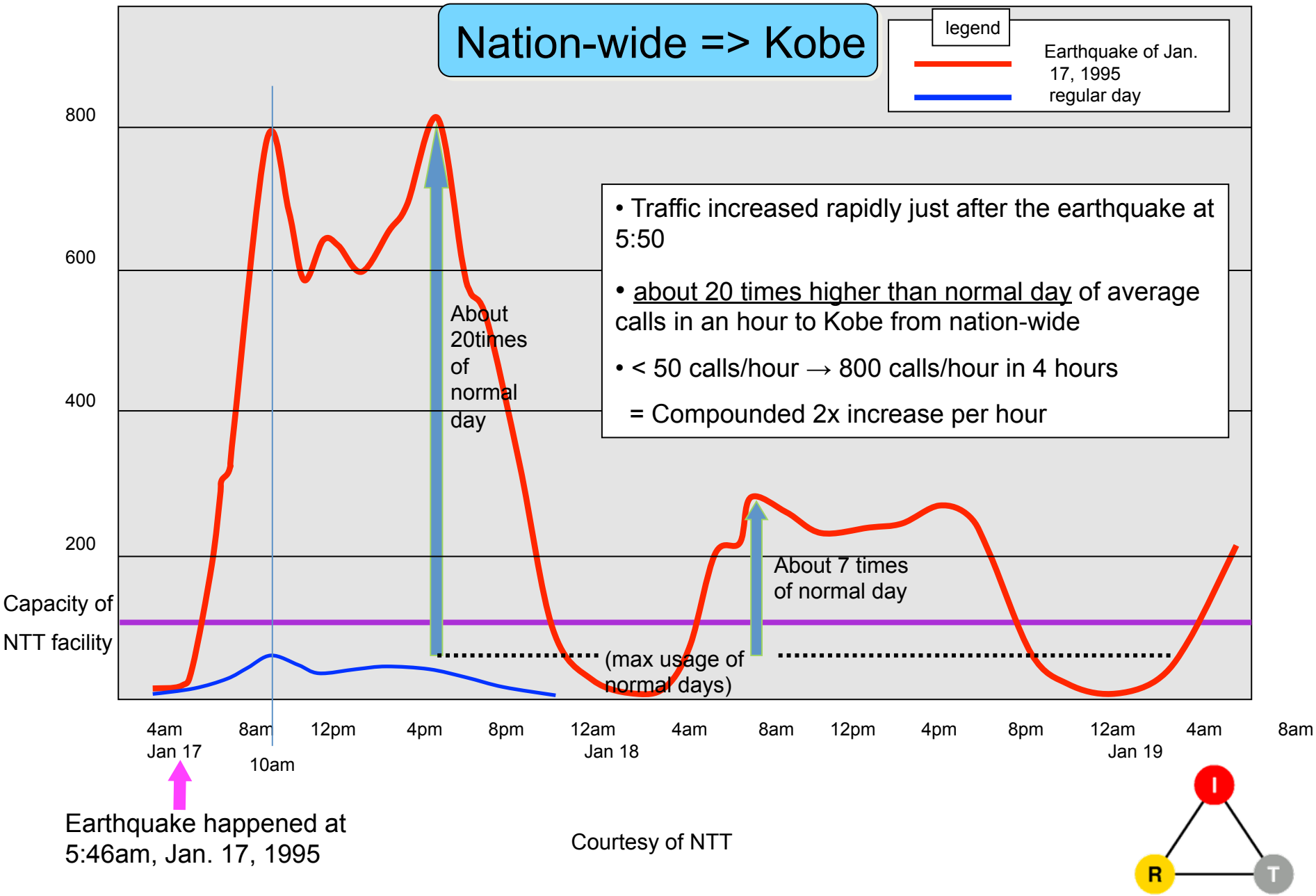
Jong Yul Kim, Henning Schulzrinne
Internet Real-Time Lab
Columbia University, USA



Contents

1. Dynamically scalable SIP services (~30 slides)
   - What is the problem we're addressing?
   - Our context and approach
   - Architecture implementation and design
   - Test results
2. Failures and how to recover from them (work-in-progress, 4 slides)
3. Messaging system (1 slide)



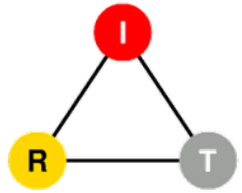

Nation-wide => Kobe
legend
Earthquake of Jan. 17, 1995
regular day
800
600
400
200
Capacity of NTT facility
About 20times of normal day
About 7 times of normal day
(max usage of normal days)
• Traffic increased rapidly just after the earthquake at 5:50
• about 20 times higher than normal day of average calls in an hour to Kobe from nation-wide
• < 50 calls/hour → 800 calls/hour in 4 hours
= Compounded 2x increase per hour
4am Jan 17
8am
10am
12pm
4pm
8pm
12am Jan 18
4am
8am
12pm
4pm
8pm
12am Jan 19
4am
8am
Earthquake happened at 5:46am, Jan. 17, 1995
Courtesy of NTT
I
R
T

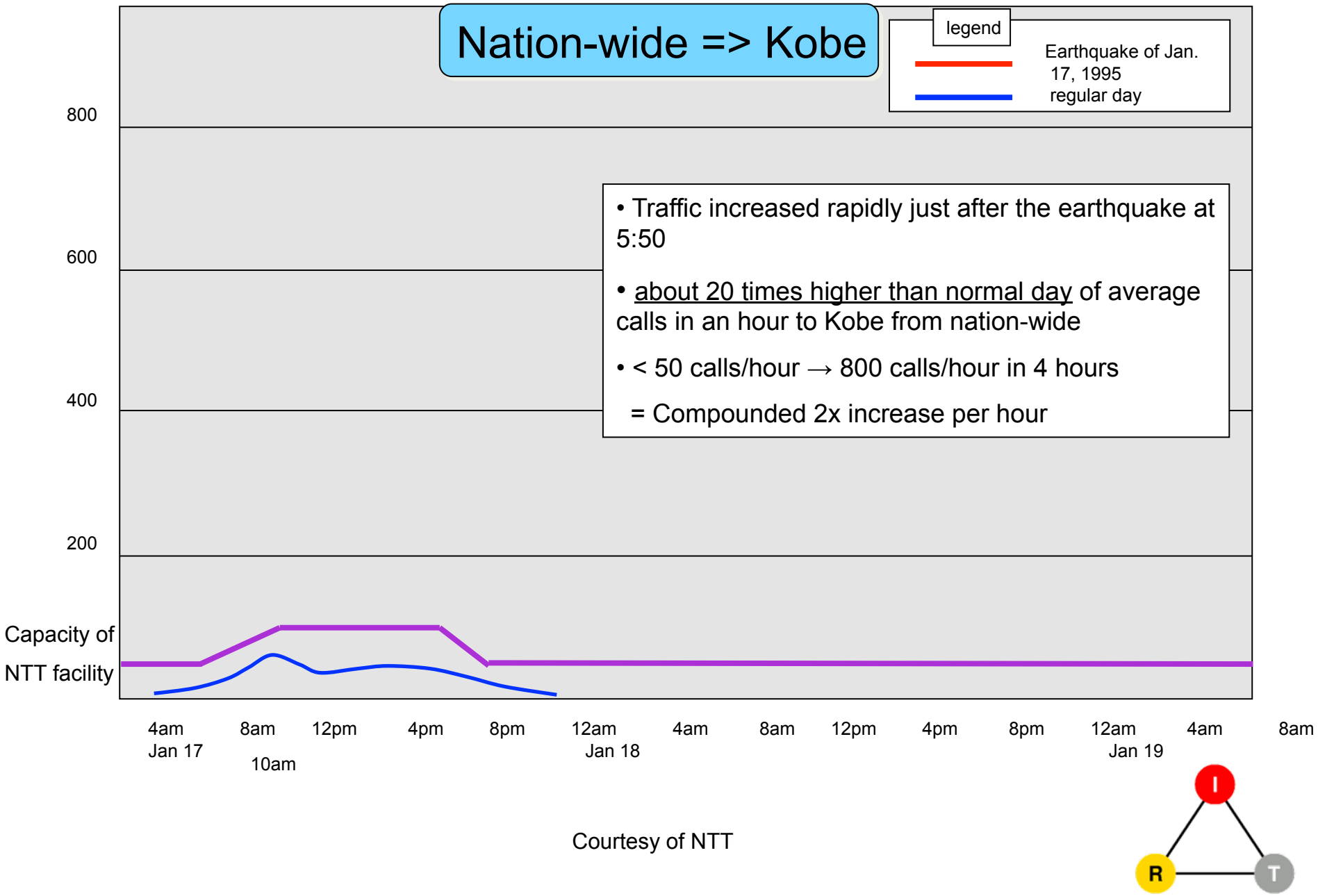
That's a problem

Capacity planning is not ideal because of:

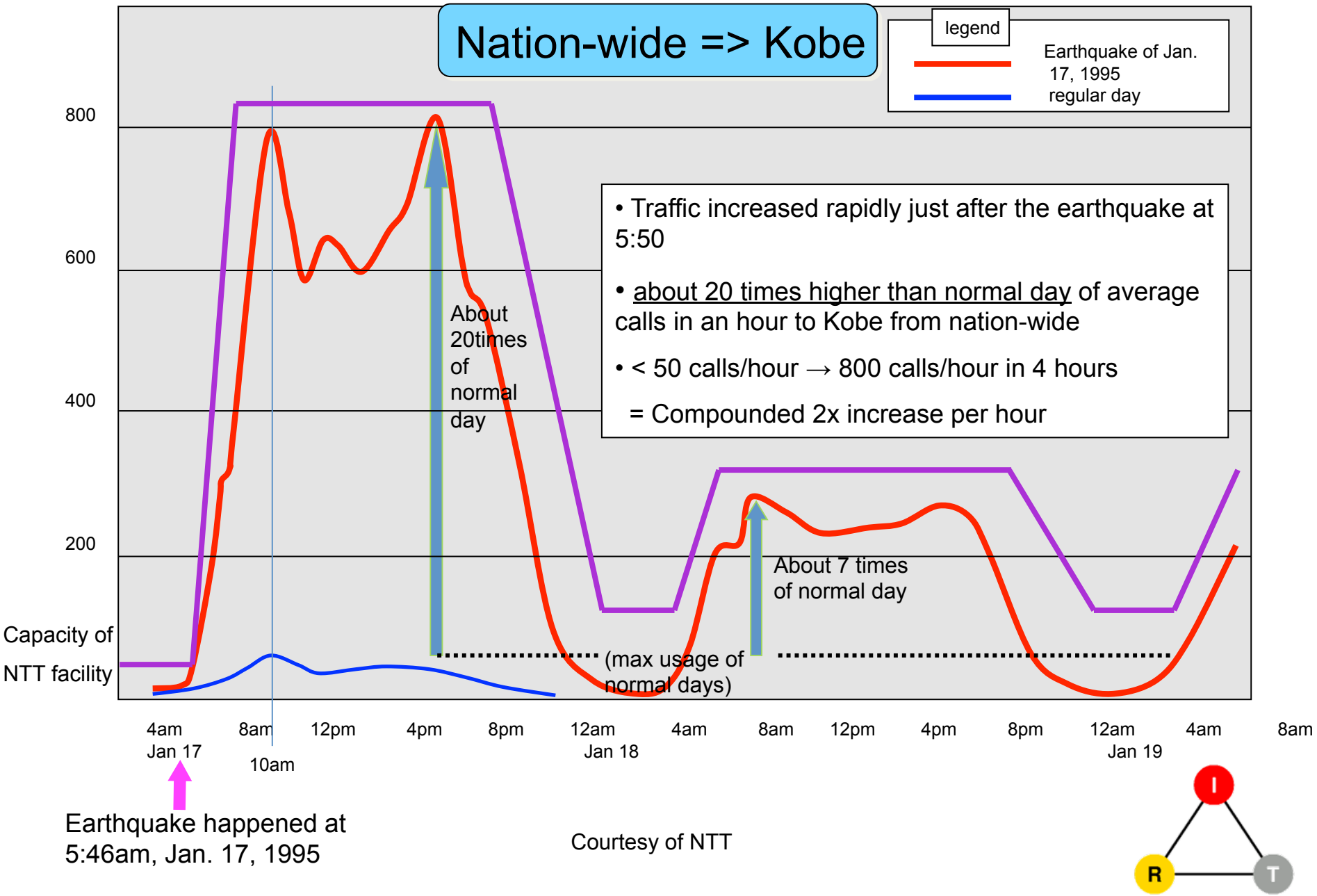
- Overprovisioning for normal operation
  - Smooth operation but some resources are idle
- Underprovisioning for unexpected events
  - Hard to predict how much traffic to expect

Can we do better?





Courtesy of NTT


Nation-wide => Kobe
legend
Earthquake of Jan. 17, 1995
regular day
800
600
400
200
Capacity of NTT facility
• Traffic increased rapidly just after the earthquake at 5:50
• about 20 times higher than normal day of average calls in an hour to Kobe from nation-wide
• < 50 calls/hour → 800 calls/hour in 4 hours
= Compounded 2x increase per hour
About 20times of normal day
About 7 times of normal day
(max usage of normal days)
4am Jan 17
8am
10am
12pm
4pm
8pm
12am Jan 18
4am
8am
12pm
4pm
8pm
12am Jan 19
4am
8am
Earthquake happened at 5:46am, Jan. 17, 1995
Courtesy of NTT
I
R
T

If we can do that…

The system can become more :

Reliable

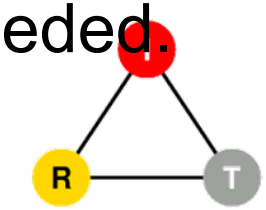
Users are not disrupted by overload situations.

Economical

There are less idle resources.

Energy efficient

Resources are used only as much as needed.



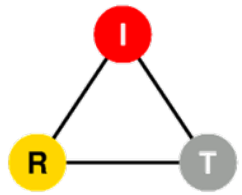
Context of our research

An internet-based voice service provider

- No PSTN interconnection
- Uses SIP for signaling

Study on the scalability of signaling plane

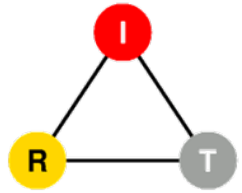
→ “How do we automatically scale the system based on incoming load?”



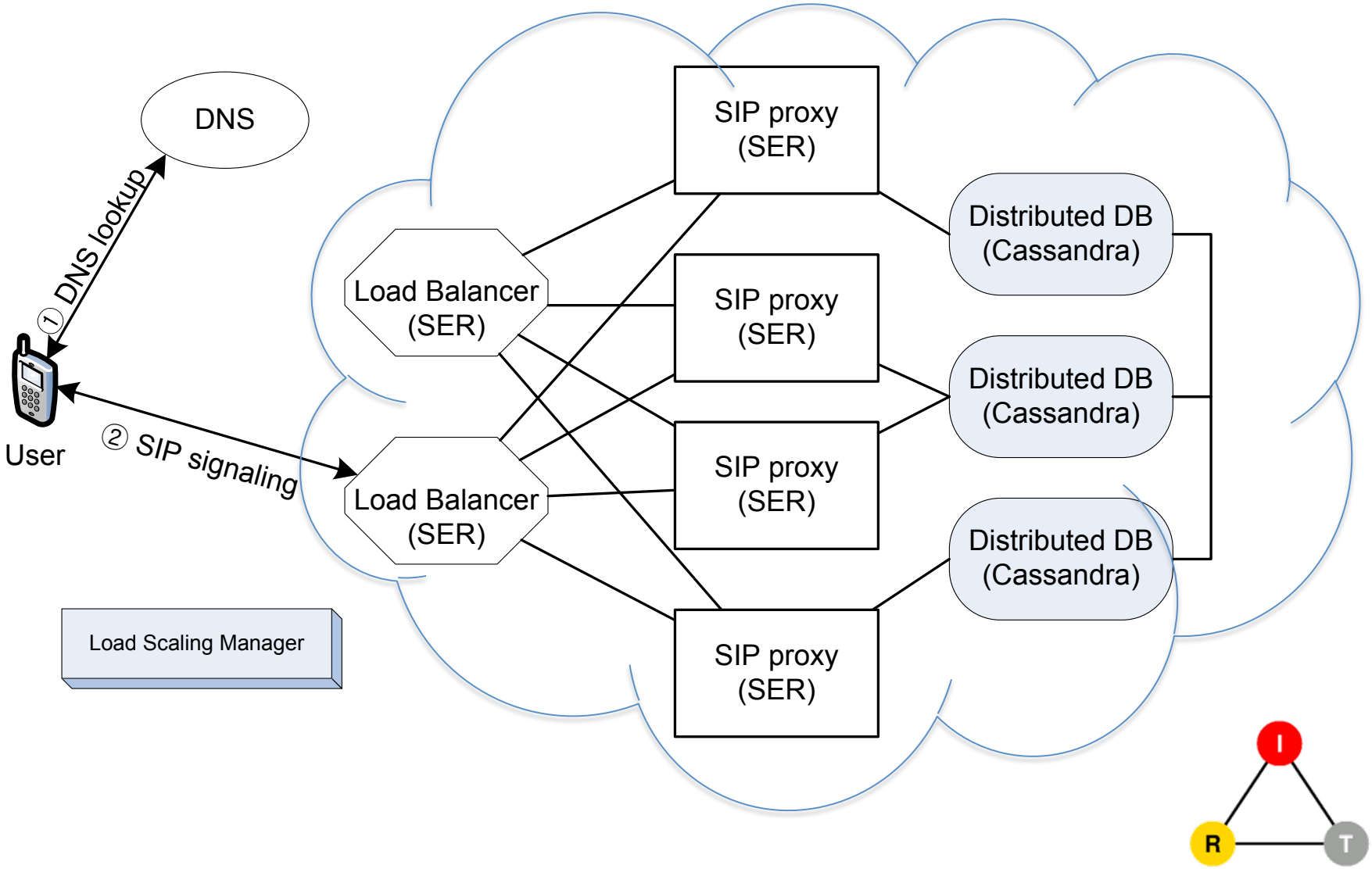
Our approach

Use Infrastructure-as-a-Service (IaaS) cloud platforms as enabling technology.

- Allows clients to add or remove VM instances on demand using the service provider's API.
- Platform users can pre-configure an appliance (OS + application) and run it as a VM instance.
- "Auto-scaling" is only for HTTP traffic.
  Let's apply it to SIP traffic as well.

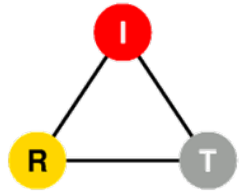


SipCloud Architecture



Load Scaling Manager (LSM)

- Has global knowledge of the cluster
- Monitors load
- Creates / terminates VM instances
- Configures VM instance as either a load balancer, proxy, or a Cassandra node
  - Handles reconfiguration of running nodes as well.
- Even if LSM is killed, cluster continues operation.
  - But does not scale.

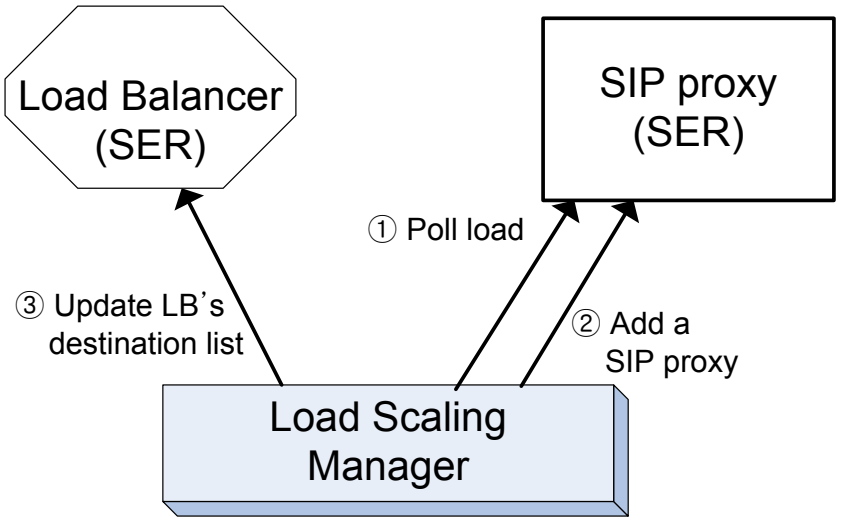


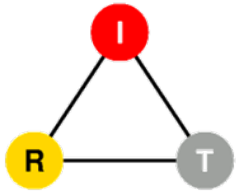
LSM adds a proxy

1 Monitor load
Load > threshold !!

2 Launch a VM
(2~3 mins),
configure proxy,
start proxy

3 Update LB's
destination list





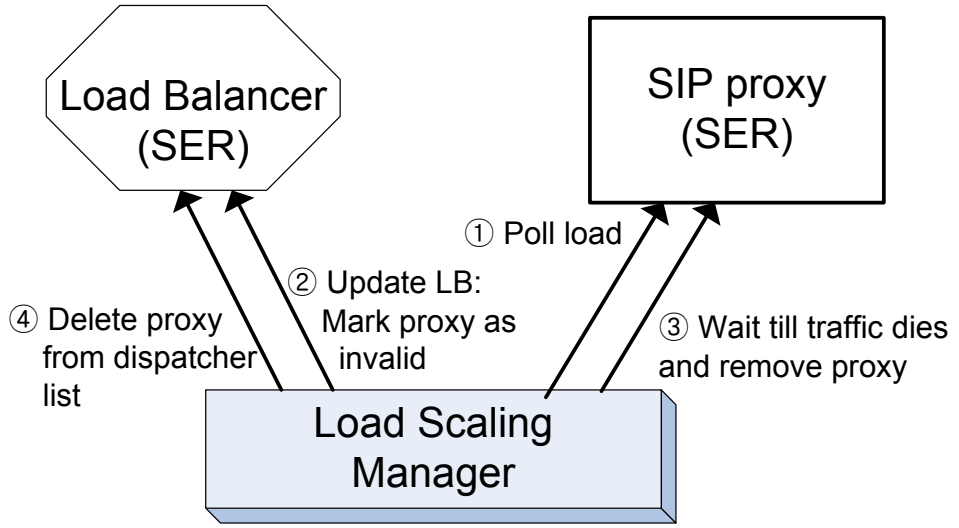
LSM removes a proxy

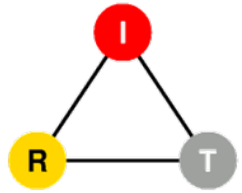
1 Monitor load
Load < threshold !!

2 Select remove candidate & update LB:
mark proxy as invalid

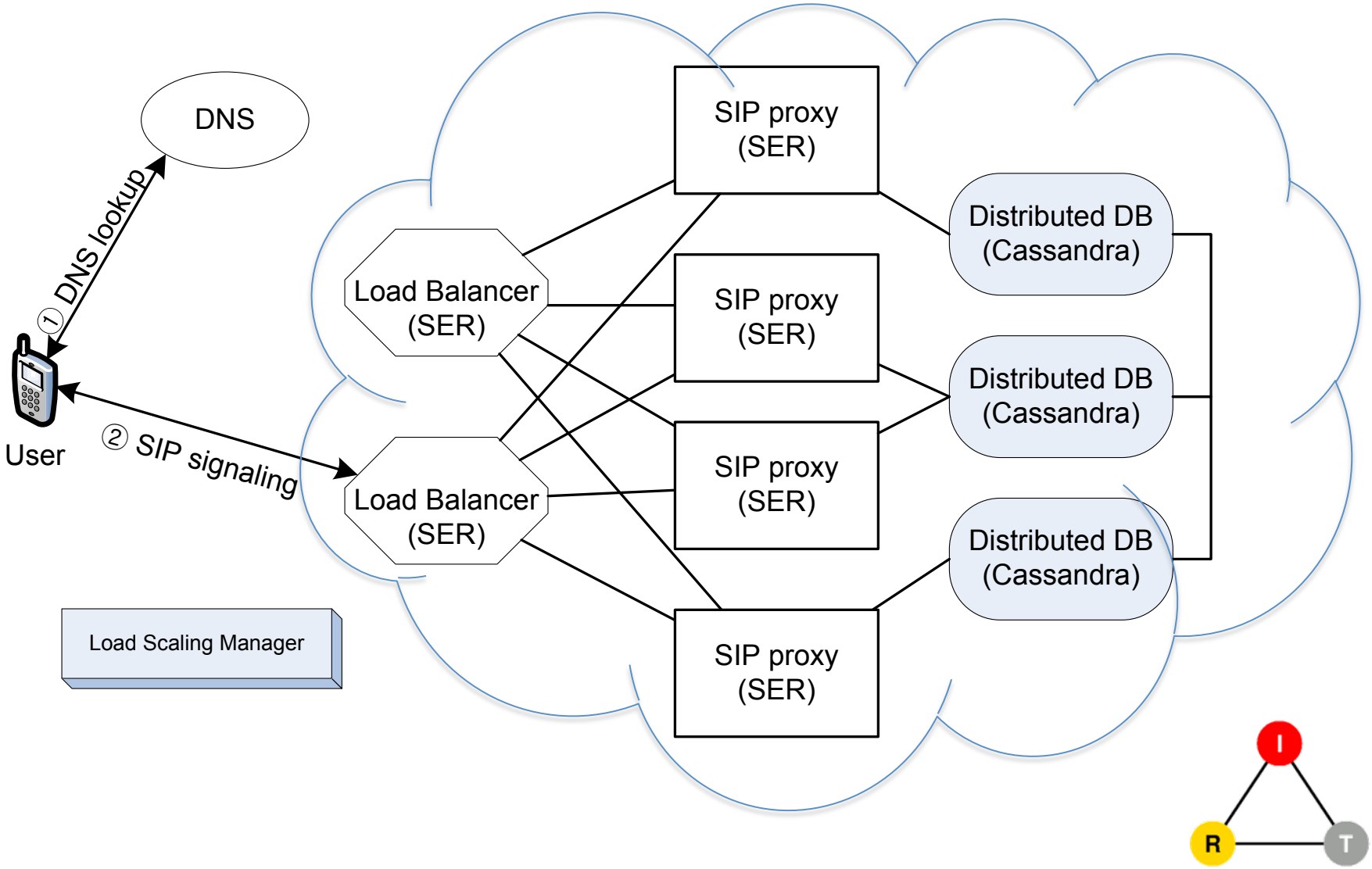
3 Terminate proxy VM

4 Update LB:
reconfigure destination file





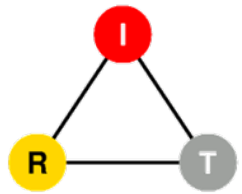
SipCloud Architecture



Designed for scalability

SipCloud is designed to utilize dynamic scalability to the fullest based on three principles:

1. Highly scalable tiers
2. Independently scalable tiers
3. Scalable database tier



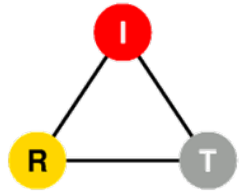
Designed for scalability

1. Highly scalable tiers

   Each tier may be able to support an unbounded* number of components if a component does not rely on another in the same tier to perform its function.

   e.g. if load balancers use a hash function to distribute load, it can operate independently from other load balancers.

   * has not been tested.



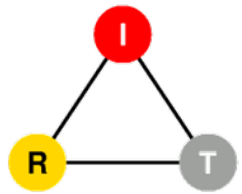
Designed for scalability

2. Independently scalable tiers

Each tier scales independently from other tiers.

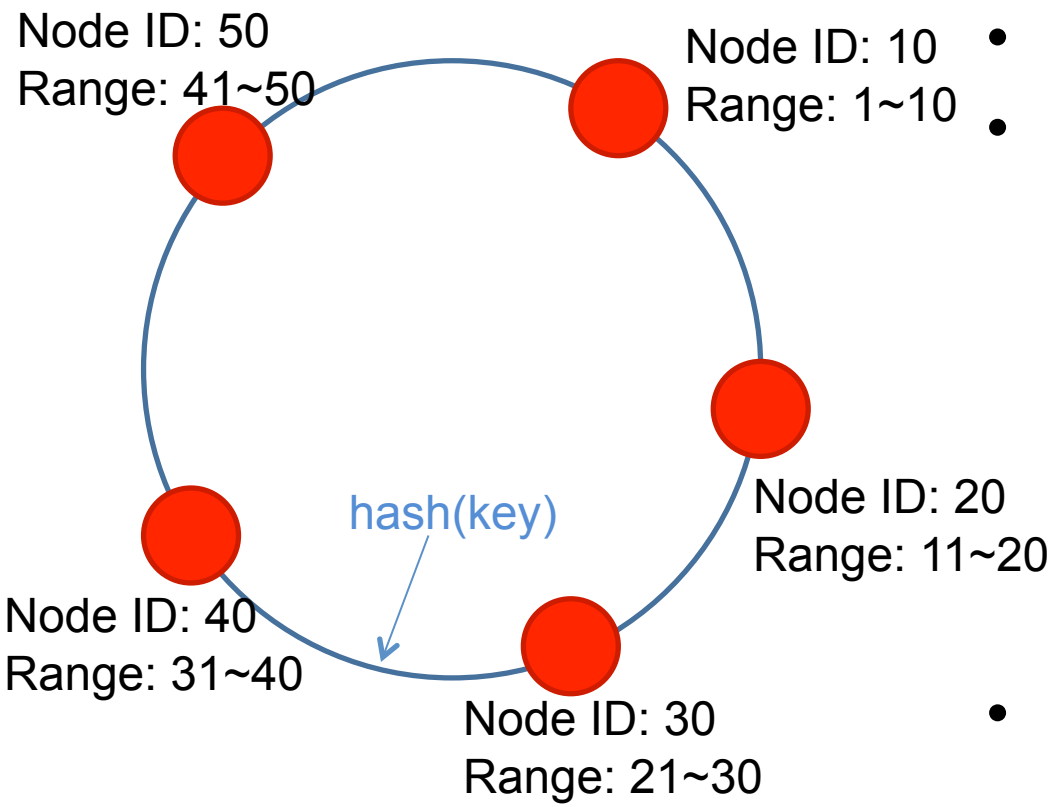
- Proxy tier scales on incoming load.
- DB tier scales on number of subscribers.

Scaling logic is simplified to a tier-local decision.

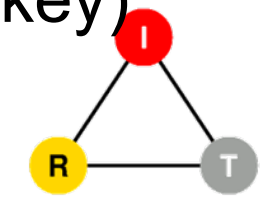


Designed for scalability

3. Scalable database



- Key-value store
- Each node
  - is a single cassandra instance
  - is a P2P node (simply add/remove node)
  - contains keys < node ID
  - replicates to N-1 successive nodes
- Key query: hash(key)



SQL vs. Key-value store

- SQL

  SELECT contact FROM db.location WHERE user='jk'

- Cassandra

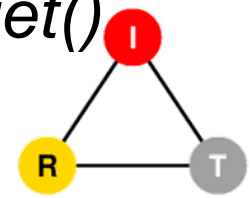
  Keyspace = 'db'

  ColumnFamily = 'location'

  Column = 'contact'

  Key = 'jk'

  – No query language. Only function call, e.g. *get()*



SQL vs. Key-value store

- SQL

  SELECT pw FROM db.credentials WHERE user='jk' and realm='cs.columbia.edu'

- Cassandra

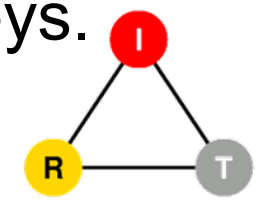
  Keyspace = 'db'

  ColumnFamily = 'credentials'

  Column = 'pw'

  Key = 'jk' → 'jk@cs.columbia.edu'

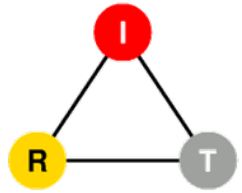
  – Design of ColumnFamily revolves around keys.



ColumnFamily for SER proxy

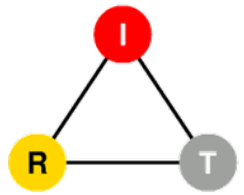
Credentials			
username@realm	password	ha1	flags

Location				
userID	AOR	contact	expires	received



Designed for scalability

Due to the three principles, it is easy to add or remove components in the SipCloud system.



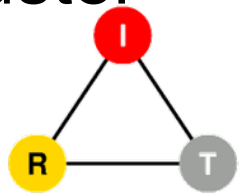
Testing dynamic scaling

On Amazon EC2, M1.Large instance

- One dual-core processor
- 7.5 GB memory
- 64-bit linux operating system

We could only perform a limited test on Amazon EC2

- 1 load balancer, 1~4 proxies, 1 Cassandra
- Up to 800 calls per second for the whole cluster



Abuse Case 13633844695

Hello,

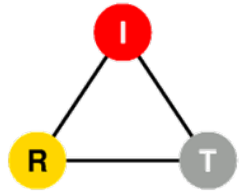
We have detected that your instance(s):i-4aeda025 have been behaving in the following way that is against our AWS Customer Agreement:

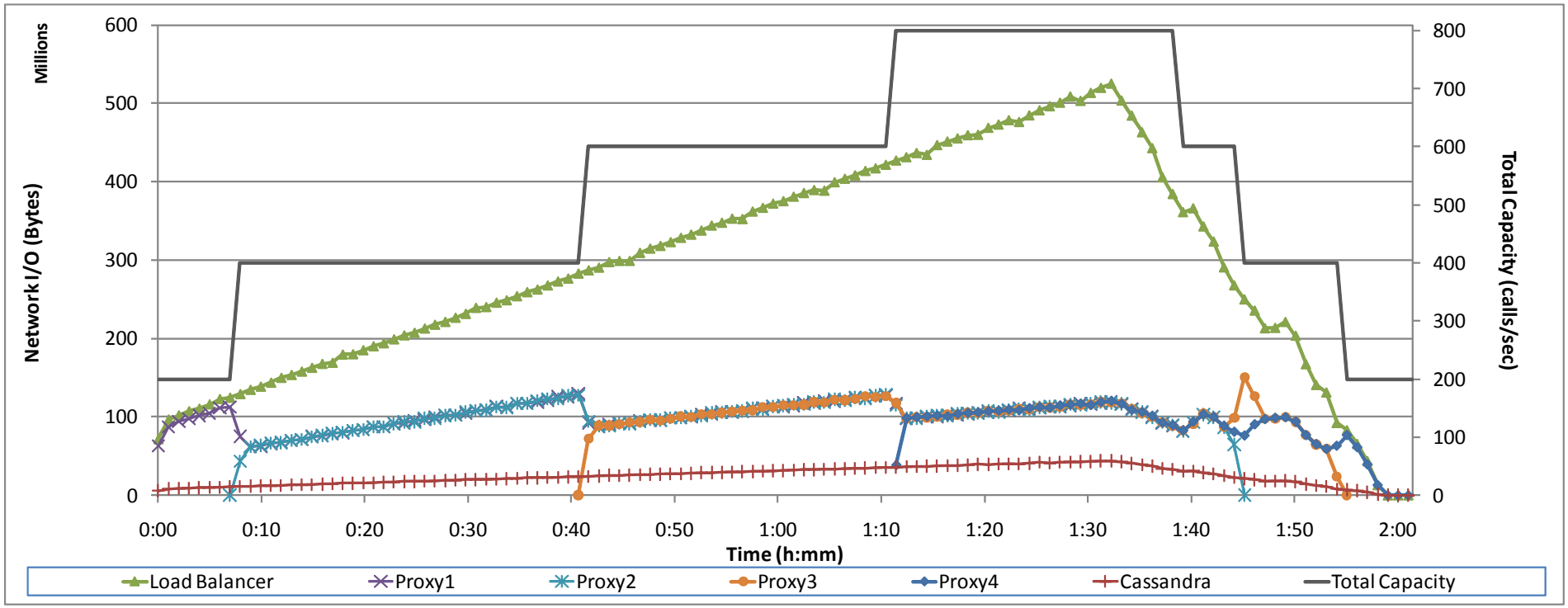
Port Scanning

Please be aware that in terms of the Web Services License Agreement http://aws.amazon.com/agreement/ if your instance(s) continue such abusive behavior, your account may be subject to termination.

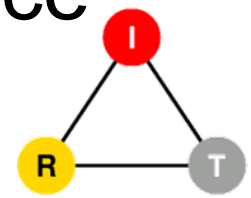
EC2 has taken the following administrative action(s) against your instance(s):

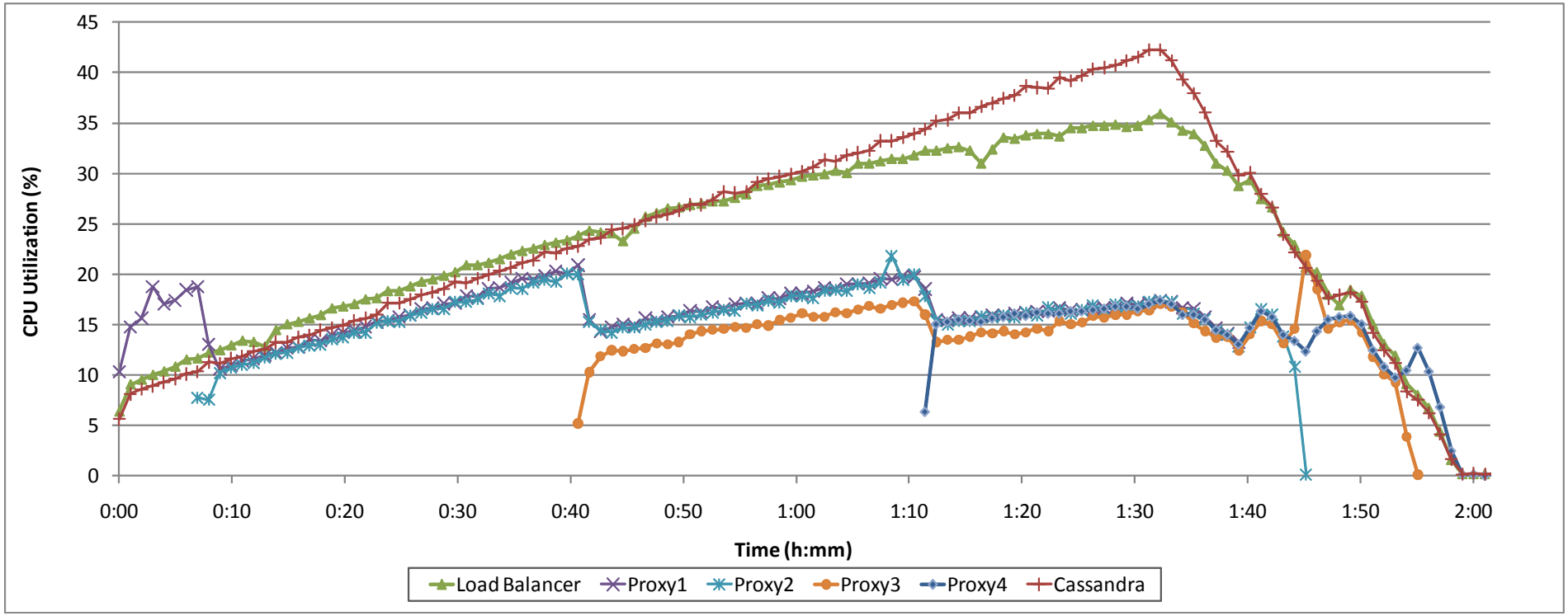
BLOCKED OUTBOUND PORT 5060



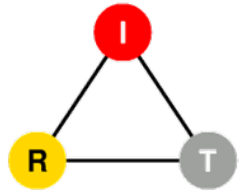


Total cluster capacity (calls per second) and Network I/O (bytes) of each VM instance





CPU utilization of VM instances



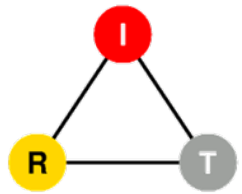
Conclusion

IaaS platforms provide dynamic scaling, which can be used by SIP service providers.

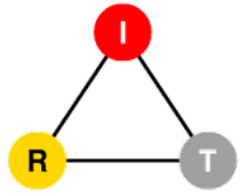
But using the feature properly requires a lot of work.

- SIP level load monitoring
- VM creation / termination
- Configuration/reconfiguration on the fly

We've had success with limited proxy scaling tests, but whether it can scale better still remains to be seen.



Current work on failures in the system



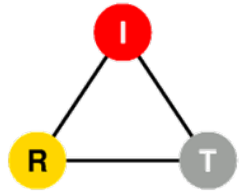
Failures

Why do they happen?

- HW failures
  - CPU, memory, disk, motherboard, network card, etc.
- SW failures
  - Parallelism (locks)
  - Missing input validation (malformed packets)
  - Cannot adapt to changes in environment (disk full)
  - Software update failures (introduce new bugs)
- Infrastructure failures
  - Network failure, DNS failure
  - Power failure

Small scale failure

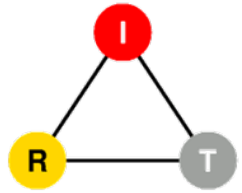
Large scale failure



Testing strategy

For small scale failures

- terminate whole VMs component-wise
  - Load balancer, SER proxy, Cassandra etc.
- collect data
  - from Load Scaling Manager's monitoring subsystem
  - Types of data:
    - Changes in VM stats: CPU utilization, network I/O
    - Changes in application: DNS record changes, proxy load changes
- deduce correlation between component failures and service failures

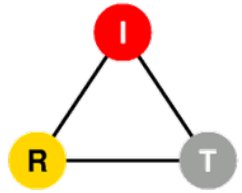


Example of a correlation result

This is a timing problem.

However,

99.999% = 5.26 minutes of down time in a year.



Possible countermeasures

Failure monitoring

- Need faster monitoring for 5-nines and beyond
- Application monitoring vs. VM monitoring

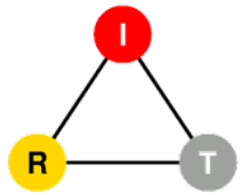
Recovery mechanisms

- Create new VM every time there's a failure.

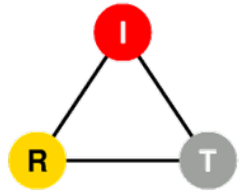
- Create new VM every time there's a failure?
  Not helpful if a malformed packet is sent.

Long term management of the system

- Deal with SW updates
- Use of heterogeneous components to build system



Scalable and reliable messaging system



Can we reuse SipCloud to build a messaging system?

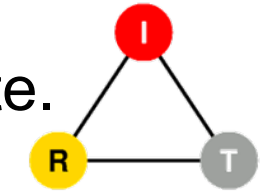
SIP INVITE

- Stateful operation
  - Real-time text (RTP)
  - MSRP (TCP)
- Will face the same advantages and problems as SipCloud for voice service.
  - Will lose state if proxy fails.

SIP MESSAGE

- Stateless operation
  - “Page mode” or “SMS mode”
- Will have better reliability as long as proxies are recovered quickly.

Either way, SipCloud can be a possible candidate.



Really, it’s the end

- We’re looking at failures and how to recover from them in a dynamically scalable system.

- A scalable, reliable messaging system based on SIP will probably face similar challenges as SipCloud.