The Power of Prediction : Cloud Bandwidth and Cost Reduction

Eyal Zohar, Israel Cidon and Osnat Mokryn

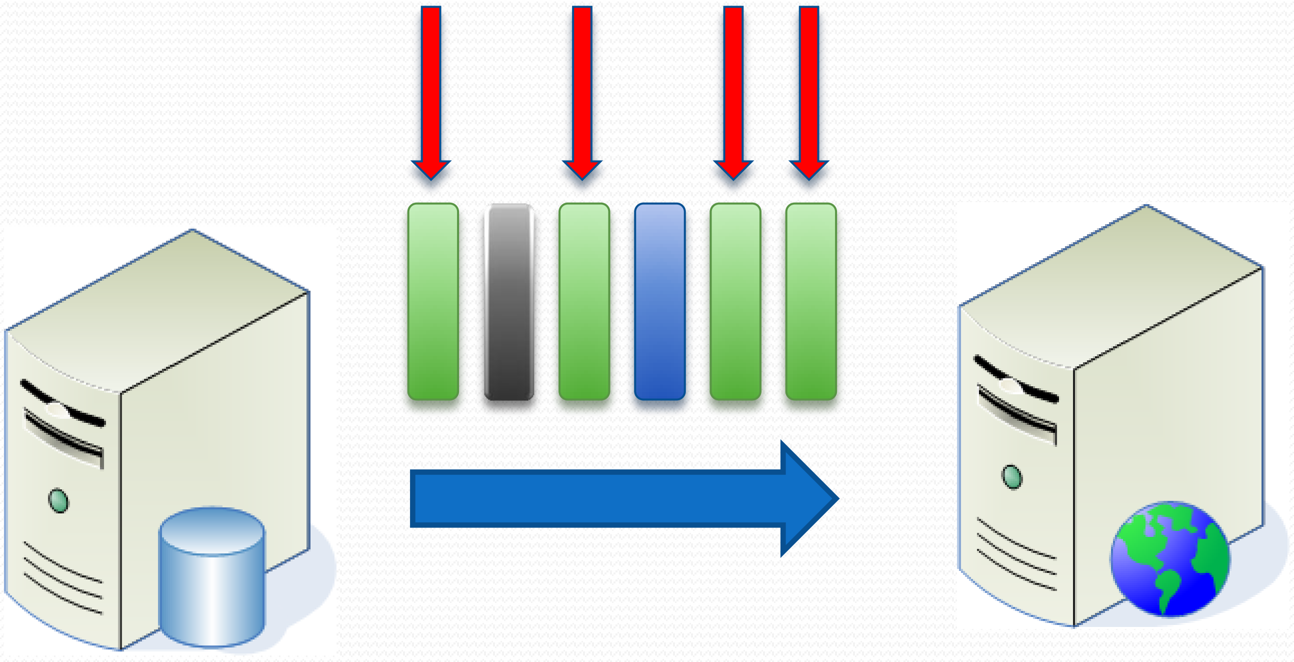
SIGCOMM 2011

Outline

- Traffic Redundancy Elimination
- Introduction to PACK
- Algorithm
- Implementation and Evaluation
- Conclusion
- CacheQuery

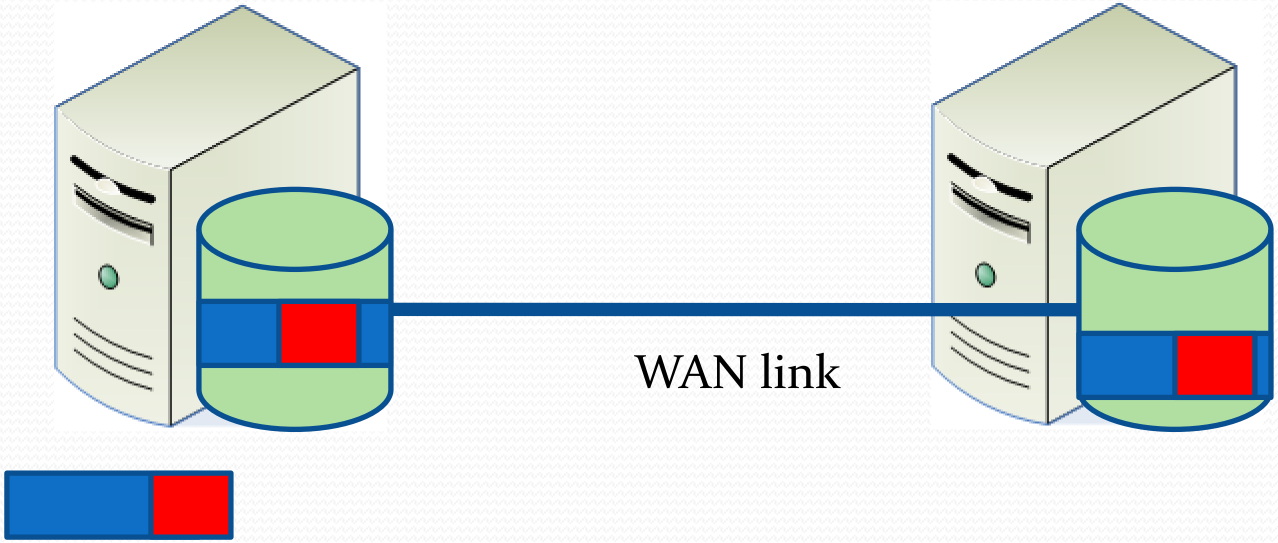
Traffic Redundancy

- Traffic redundancy from transmitting similar data.



Traffic Redundancy Elimination

- Sender-based TRE



Traffic Redundancy Elimination

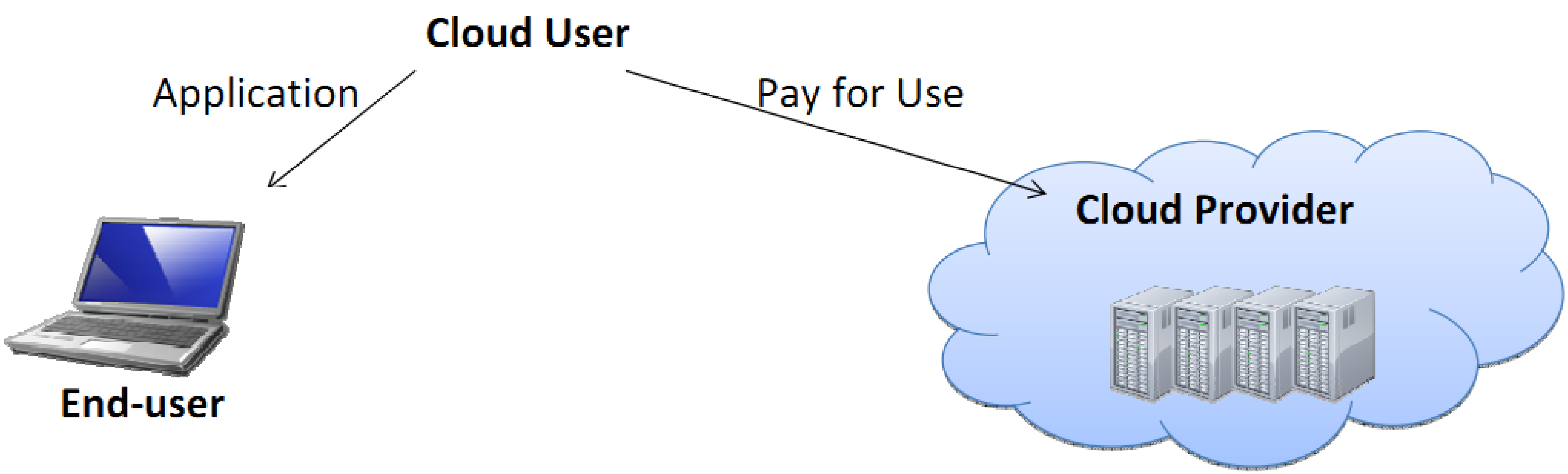
- Sender-based TRE
  - Middle Box
  - EndRE (NSDI' 10)

Sender-based TRE

- Weakness
  - Inefficient on **mobile** environment
  - **Computation** cost on **sender side**
  - **Synchronization** between sender and receiver

TRE Importance

- Service on cloud
  - Higher traffic → Higher cost
  - Incentive to use TRE

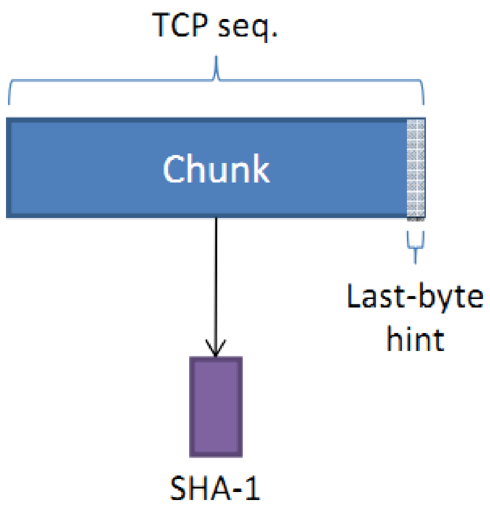


Introduction - PACK

- Predictive ACK (PACK)
  - **Receiver-based** end-to-end TRE
  - Redundancy detection by the **receiver**
  - Tries to match incoming chunks with a previously received **chain**
  - Send the **predictions** of future data to the sender

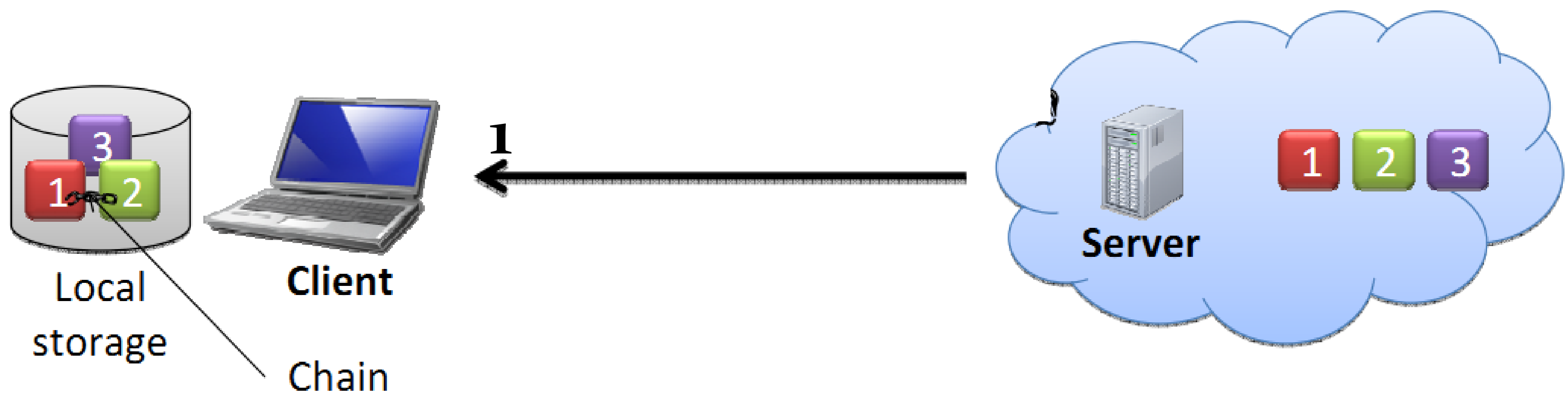
PACK - Receiver

- Chunk store
  - **Chunk**
  - **Meta data** : signature、hint、 pointer



PACK - Sender

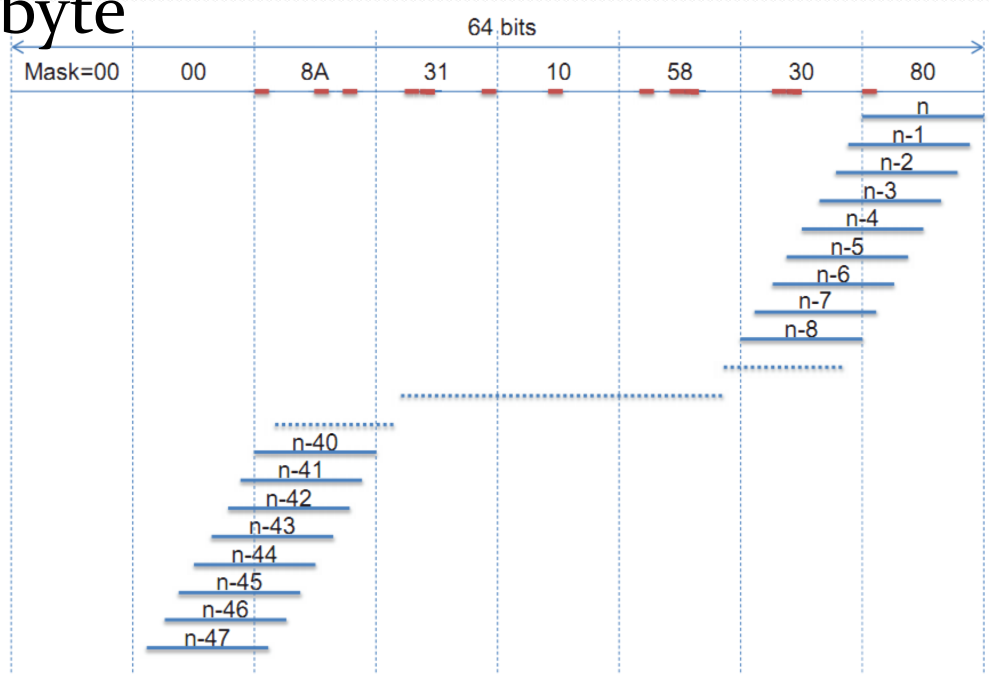
- Compares the **hint** with the last-byte to sign
- Upon a hint match it performs the expensive **SHA-1**



Operations

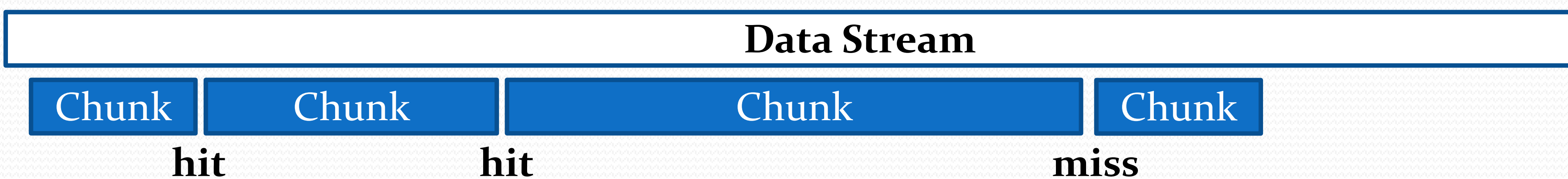
Chunking Algorithm

- Choose chunk's **entrance point** (anchor)
- 8 bytes Mask
- 48 bytes Window
- X-OR for each incoming byte



Optimizations

- **Adaptive receiver virtual window**
  - **Increase** window size with each prediction **success**
  - **Reset** window size while **miss** prediction
  - Tradeoff between **potential gain** and **recovery effort**



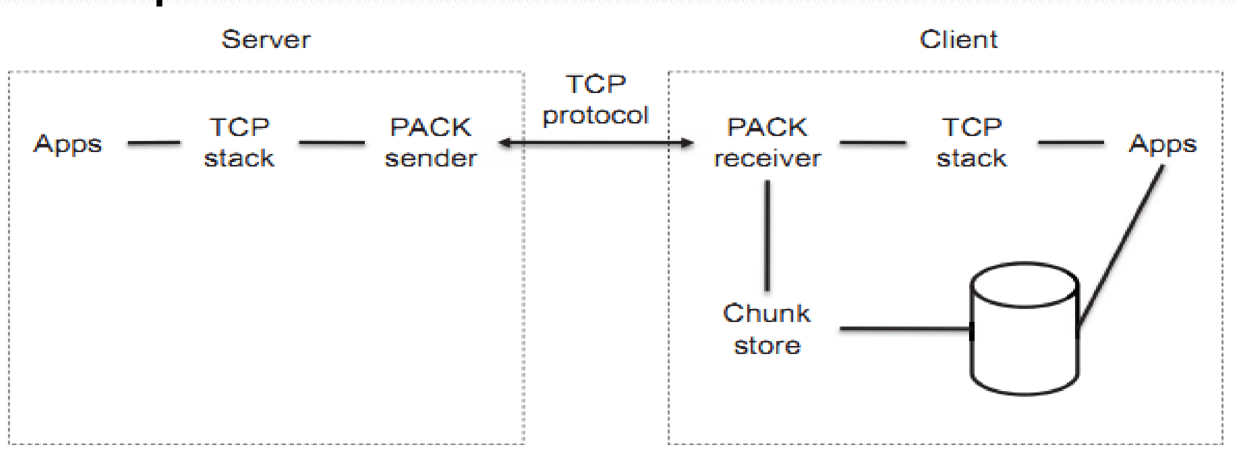
Optimizations

- **Hybrid Approach**
  - **Less efficient** if changes in the data are **scattered**
  - Report **dispersion** to sender
  - Start **sender-driven** operation if sender has enough resource

  - **Smoothing function** $D \leftarrow \alpha D + (1 - \alpha) M$
    - From **0** to **255** (long smooth)
    - **M** : set to **0** while chain **break**, and **255** otherwise

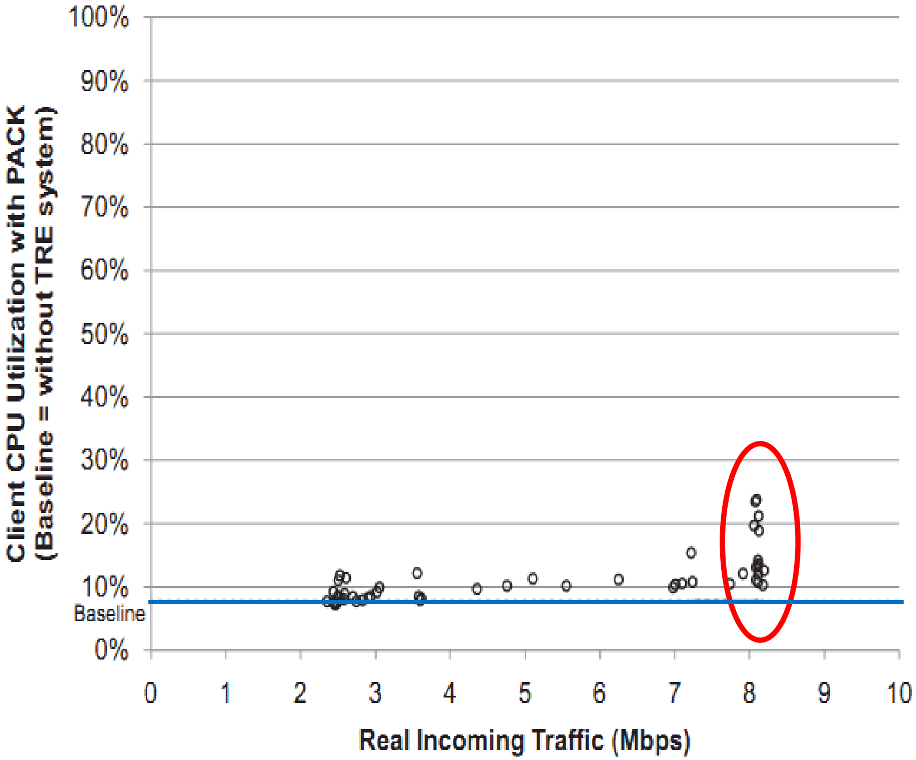
Implementation

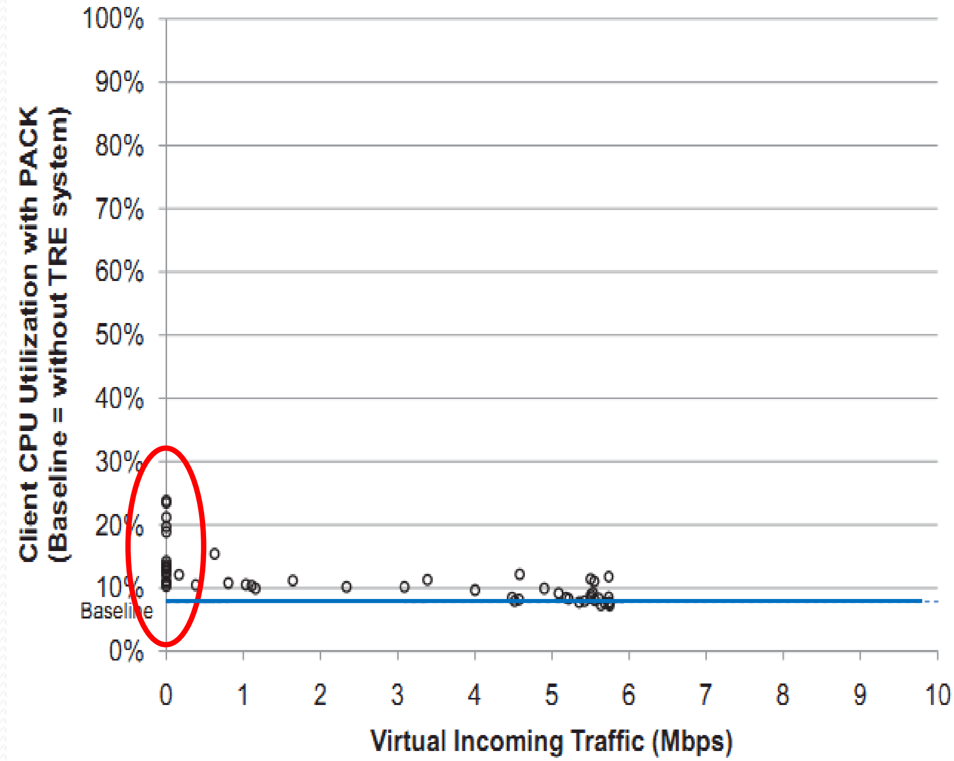
- **Protocol** is embedded in the **TCP Options field**
- Average **chunk size** : 8 KB
- Run on Linux with **Netfilter Queue**
- **Additional overhead**
  - 0.1% storage for meta-data
  - 0.15% bandwidth for predictions



Client CPU cost

- **No-TRE** avg. CPU : 7.85%
- **PACK** avg. CPU : 11.22%





Evaluation

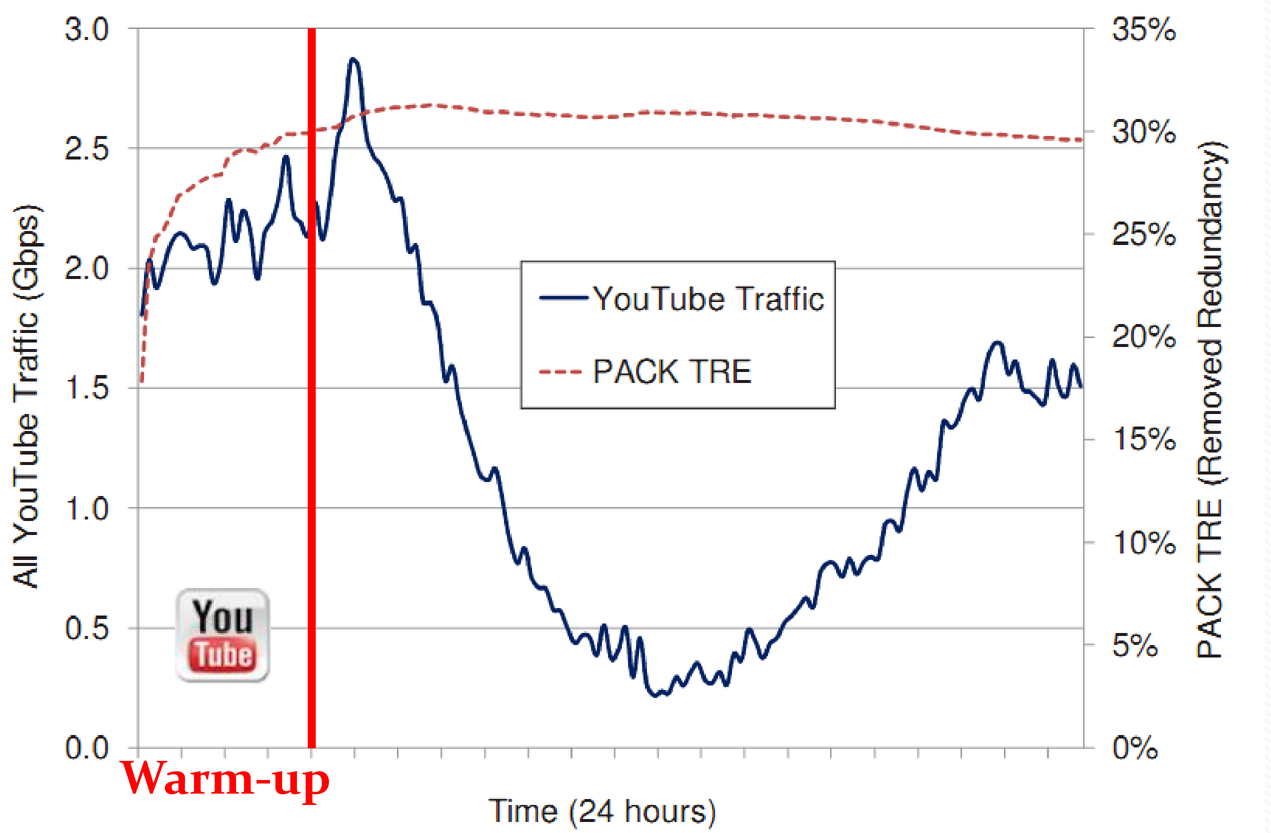
- **Traffic traces**
  - **Video traces** captured at a major **ISP**
  - Traffic from a popular **social network** service
- **Static data sets** of real-life workloads
  - Linux source
  - Email

Evaluation – Video trace

- 24 hours from ISP's 10 Gbps router
- Filtered **YouTube** traffic, total **1.55 TB**
- Total **40k** clients

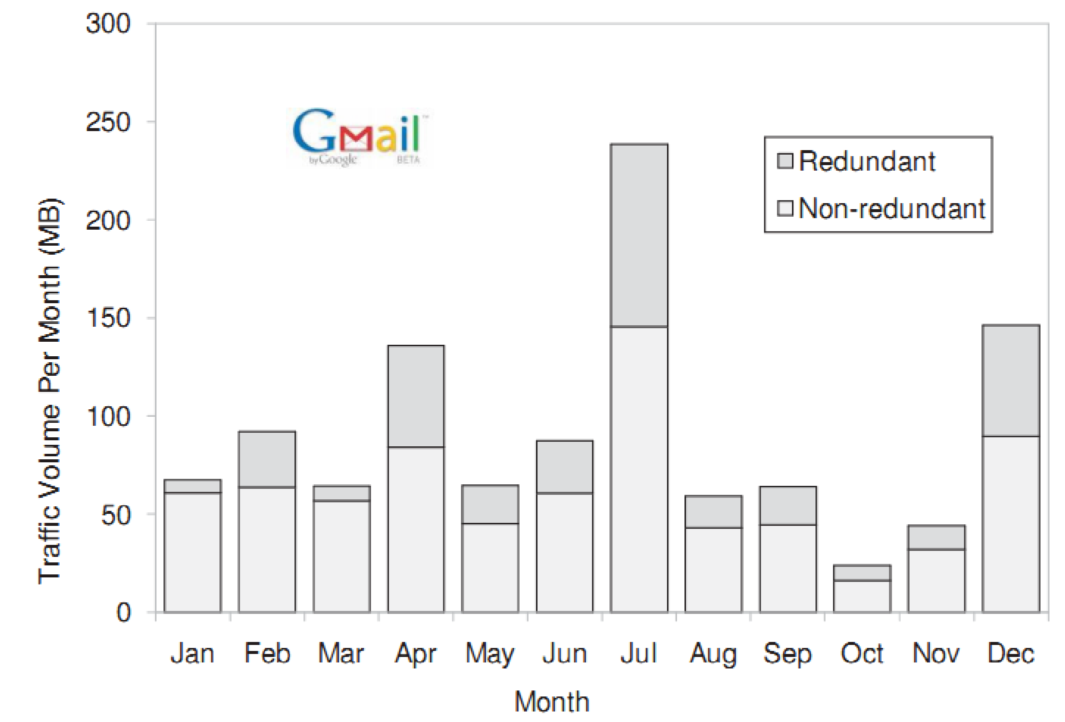
Evaluation – Video trace

- Users very often download the same video or parts
- **30%** end-to-end redundancy



Evaluation – Gmail

- Gmail account with 1,000 Inbox messages
- Found 32%static redundancy
  - higher when messages are read multiple times



Estimated Cloud Cost

- YouTube traffic trace
- An array of such servers, for each
  - **Outputs** up to **350** Mbps, **600** concurrent **clients**
  - Control **computation power** between 0.25 and 0.5
- Amazon EC2
  - Traffic : Server-hours cost ratio = 7 : 3

	No TRE	PACK	Server-based
Traffic volume	9.1 TB	6.4 TB	6.2 TB
Traffic cost reduction		30%	32%
Server-hours cost increase		6.1%	19.0%
Total operational cost	100%	80.6%	83.0%

Conclusion

- Current TRE solutions may not **reduce cloud cost**
- **Minimizes processing costs** induced by TRE
- Suitable for server migration and client **mobility**

Weakness

- No receiver **storage** information
  - PACK will cause to receiver maintains huge data
  - Maybe give an example of **resource-constrained devices**, such as mobile phones
- Less efficient caused by **sporadic changes**
  - Assume sender send the **same** and **long-term** stream
  - Like **video**, **mail** and **linux kernel header**

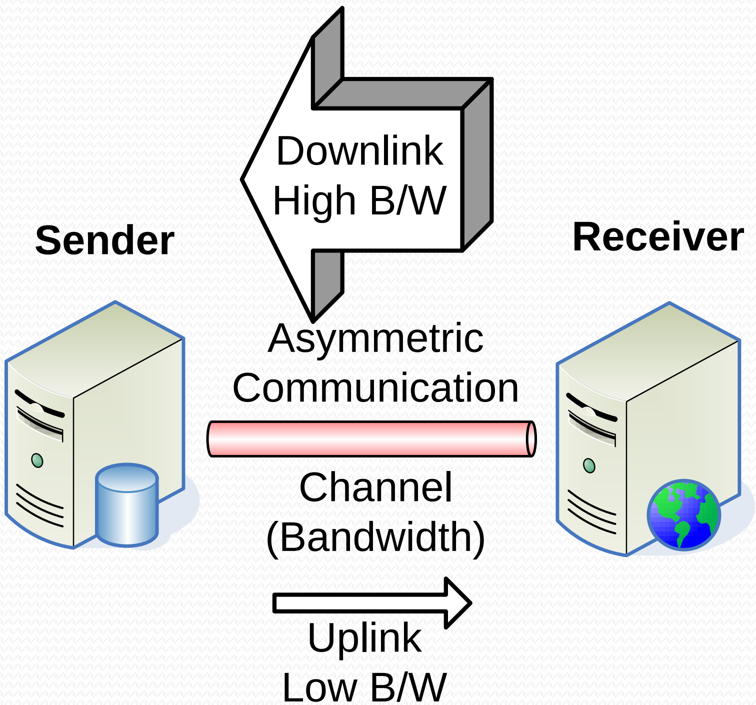
Problem Statement and Solution

- **Increasing uplink** rate in **asymmetric communications**
  - by capitalizing the otherwise wasted downlink bandwidth and/or receiver capability.

- **Asymmetric Redundancy Elimination** (CacheQuery)
  - on top of TCP
  - increases the uplink rate from **multiple senders** to **one or more receivers**.

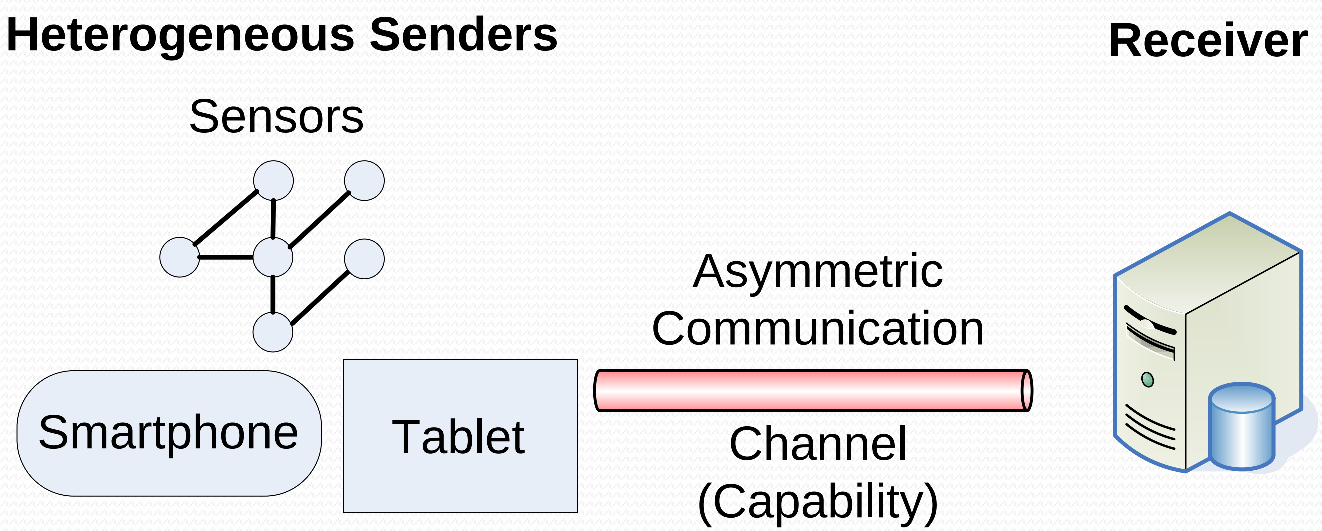
What We Consider ?

- **Bandwidth asymmetric channels**



What We Consider ?

- **Capability asymmetric channels**

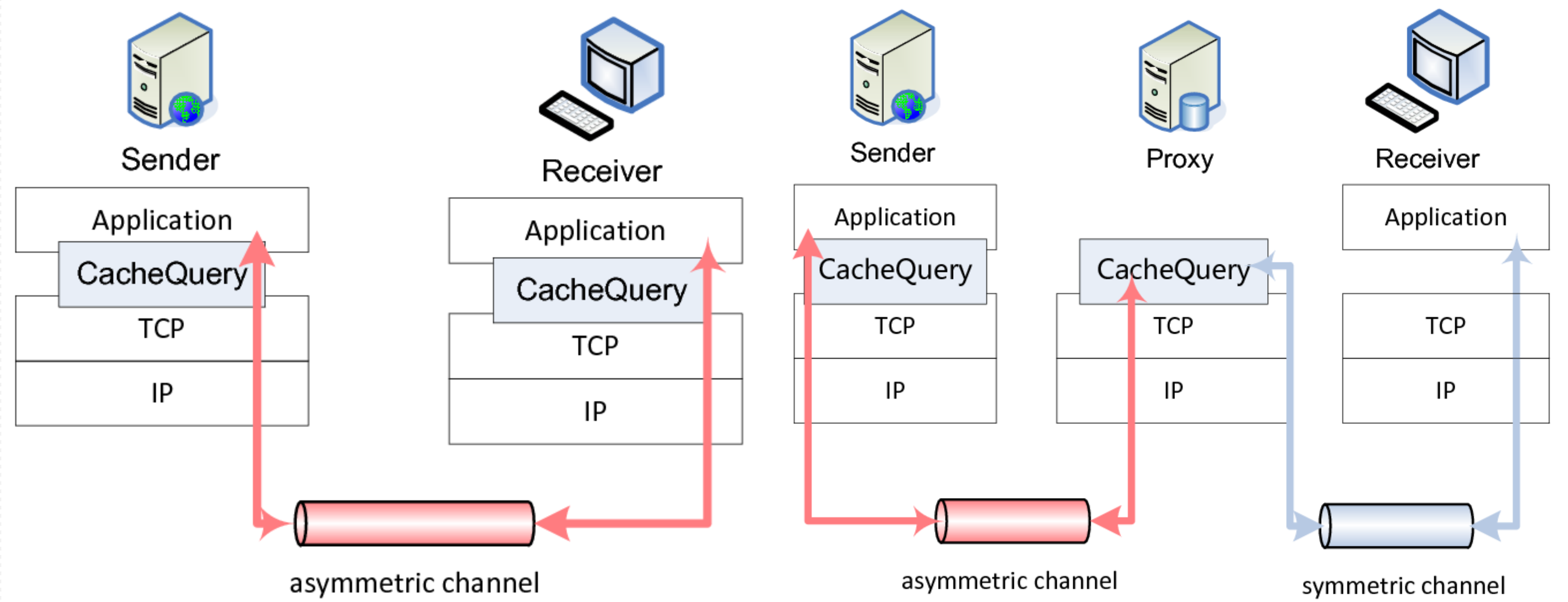


- Slower CPU
- Smaller memory
- Battery powered

- Faster CPU
- Larger memory
- Power line powered

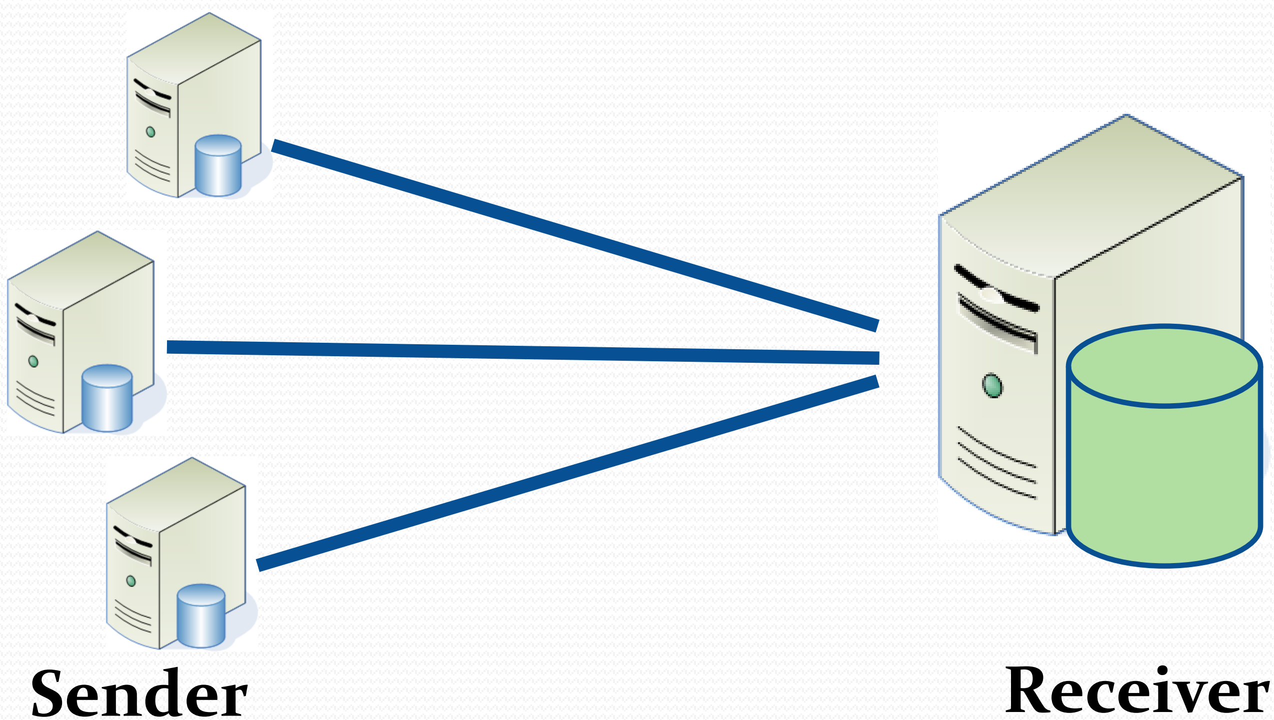
CacheQuery Can be Deployed on

- **two end-systems**
- **network proxy**



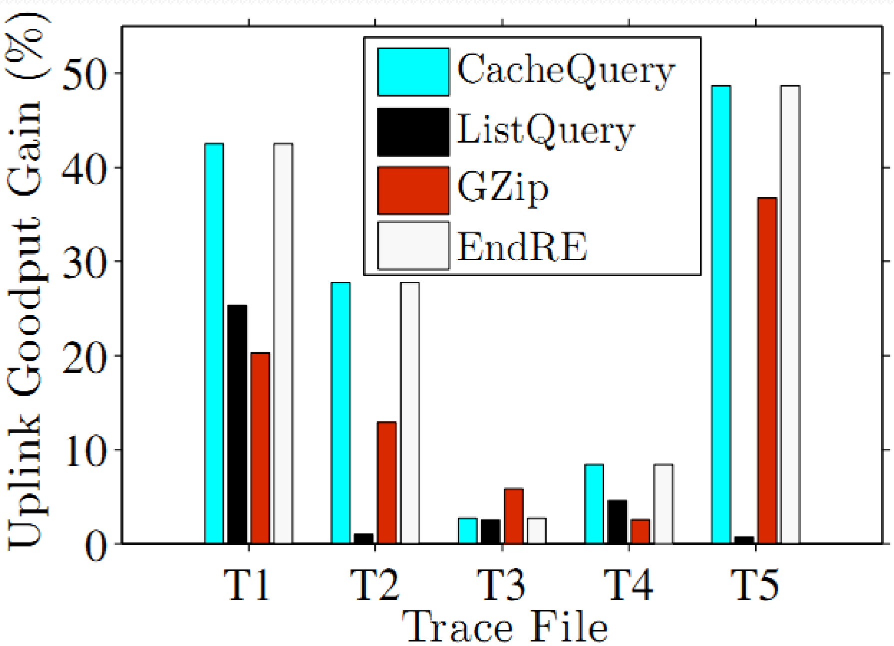
CacheQuery

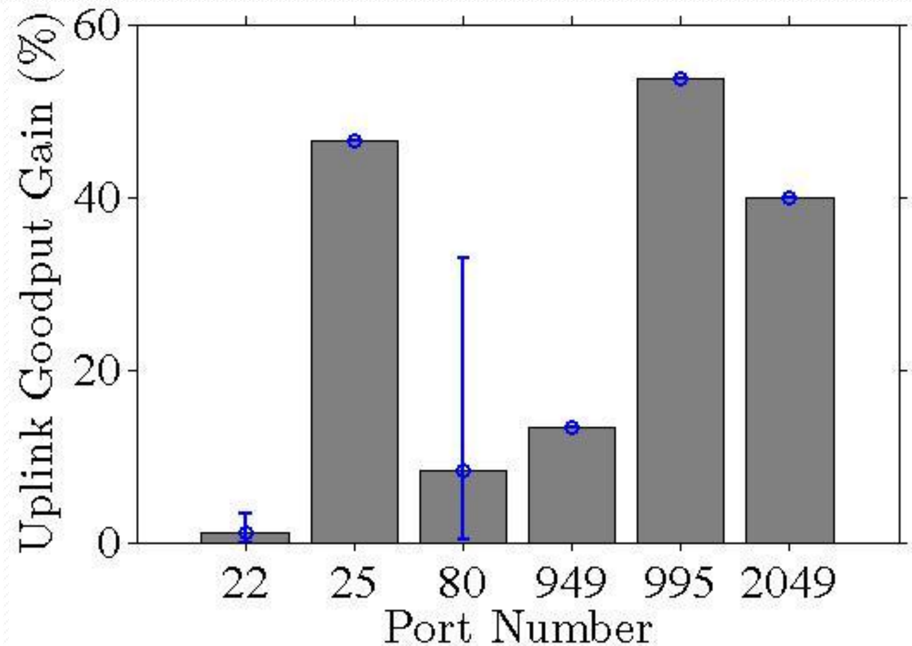
- Sender and receiver maintain their own **scalable store**
- The data store will be updated periodly



Simulation Result

- Significant outperforms ListQuery [1], comparable to CacheQuery
- Diverse goodput gain among traces and protocols
- Current traces are not sufficient for concrete conclusions





[1] C. Trang, X. Huang, and C. Hsu, "Pushing uplink goodput of an asymmetric access network beyond its uplink bandwidth," ICC'12

Future Work

- Collect trace file from NCTU dorm router to get more concrete results
- Implement PACK
  - compare performance in large files

Reference

- PACK slides (some figures)
  - http://conferences.sigcomm.org/sigcomm/2011/slides/s86.pdf
- Redundancy in Network Traffic: Findings and Implications slides (some figures)
  - https://pages.cs.wisc.edu/~ashok/re-meas.pptx