Virtualizing the Access Network via Open APIs

Vijay Sivaraman, Tim Moors (UNSW)
Hassan Habibi Gharakheili, Denis Ong (UNSW)
John Matthews, Craig Russell (CSIRO)
Acknowledgement: Josh Bailey (Google)

Overview

- This paper is about service quality
- … that we have been trying for 20 years
- … with rather limited success
  - Technologies: ATM, RSVP, IntServ, DiffServ, …
- But indulge me one more time
  - Maybe SDN can add some magic dust?
- Focus less on technology
- and more on ecosystem, interfaces, architecture
  - Contentious areas: two-sided revenue, net neutrality

Service Quality: User Perspective

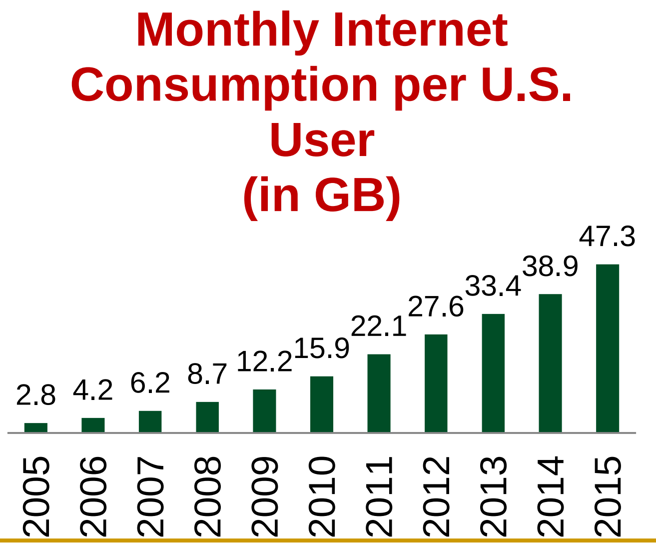
- Demanding, impatient, short attention span
  - E.g. Streaming Video [Sigcomm 2011, IMC 2012]:
    - Each second of startup delay causes 5.8% abandonments
    - Rebuffering delay of 1% reduces viewing time by 5%
- Growing number of household devices
  - Computers, tablets, smart-phones, TVs, IoT, …
  - Increased peak-load and congestion on access link
- Yes indeed users want better quality!
  - But not really willing to pay more
  - How much control over quality do users want?

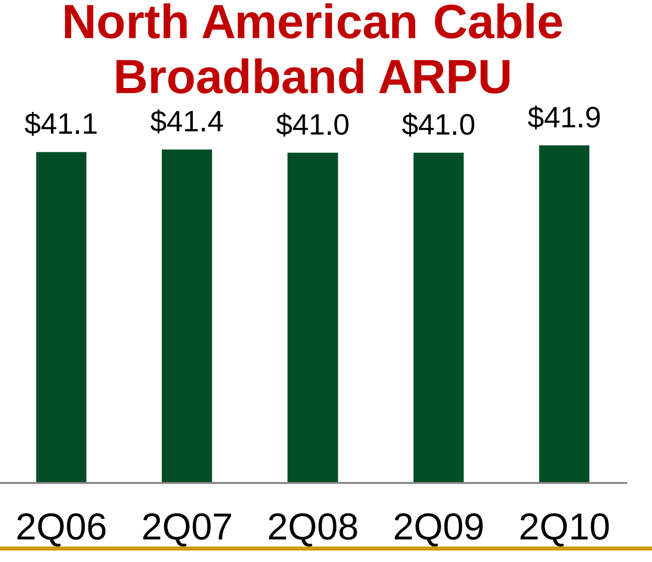
Service Quality: Content Provider Perspective

- Subscription or ad-based revenues
  - Seriously impacted by user abandonment and reduced engagement
- Yes indeed CPs want better quality!
  - Are they willing to pay for it?
  - How do they exercise control over quality?
  - Paid peering or other arrangement?
- Quality requirements of diverse services:
  - Streaming video: bandwidth assurance
  - Browsing, interactive voice/video: low latency/jitter
  - Gaming, Bulk transfers: low loss

Service Quality: ISP Perspective

- Hard time keeping up with traffic growth
  - Exponential traffic growth; flat revenue per user
- Access network bandwidth is expensive!
  - Average downlink speed: 8.7 Mbps (US), 3.3 Mbps (world)
- Incentive to improve quality?
  - User retention? Two-sided business model (revenue from CPs)?





Everyone wants service quality, but …

- Who controls it?
  - ISP: implements machinery, but
    - Transparency? Neutrality?
  - User: ultimate recipient of service, but
    - What knobs? Complexity?
  - CP: knows service characteristics but
    - How to signal requirements? What are the assurances?
- Who pays for it?
  - ISP: need to cover costs, generate revenue
  - User: cost sensitive, unlikely to pay
  - CP: paid peering? “selective” not “wholesale”?

Our proposal: SDN-driven Virtualization

- Service quality control exposed via “APIs”
  - Create dynamic on-demand “slices” in the network
  - Central “brain” executes network-wide capability
    - No protocol peering (in fact no peering needed at all)
    - Optimal resource partitioning, rapid computation
  - Selective (rather than bulk) control over quality
- Architectural decisions:
  - APIs open for (any) content provider
  - Users given single knob to control participation level
  - Only (pooled) access links partitioned

Use-cases

- QoE for streaming video (e.g. YouTube, NetFlix):
  - Network API for flow bandwidth assurance
    - Flow-id, bandwidth requirement, duration
  - User requests video → Server calls network API
    - Negotiation to agree on bandwidth, duration, price
  - Video ends / user aborts → bandwidth cancelled or expires
- Elastic bulk transfer (e.g. Software upgrades, P2P)
  - Network API for delay elasticity
    - Flow-id, file size, delay tolerance
  - Allows network to better schedule resources
    - Shifting load to lull periods → lower cost
- Multiple access paths (peak demand off-load)
  - WiFi pooling in high-density areas with coverage overlaps
  - Choice of physical paths to reach device (network virtualisation)

Benefits for ISP

- Monetization opportunity
  - Two-sided business model, per-stream revenues
- Open API: Any CP can use it
  - No back-room business arrangements needed
- Explicitly learns application characteristics
  - Reduce DPI costs
- Can protect sensitive details
  - E.g. Network topology, congestion state
- Free to innovate:
  - Algorithms for routing/slicing (e.g. WiFi pooling)
  - Pricing models, e.g. congestion-based

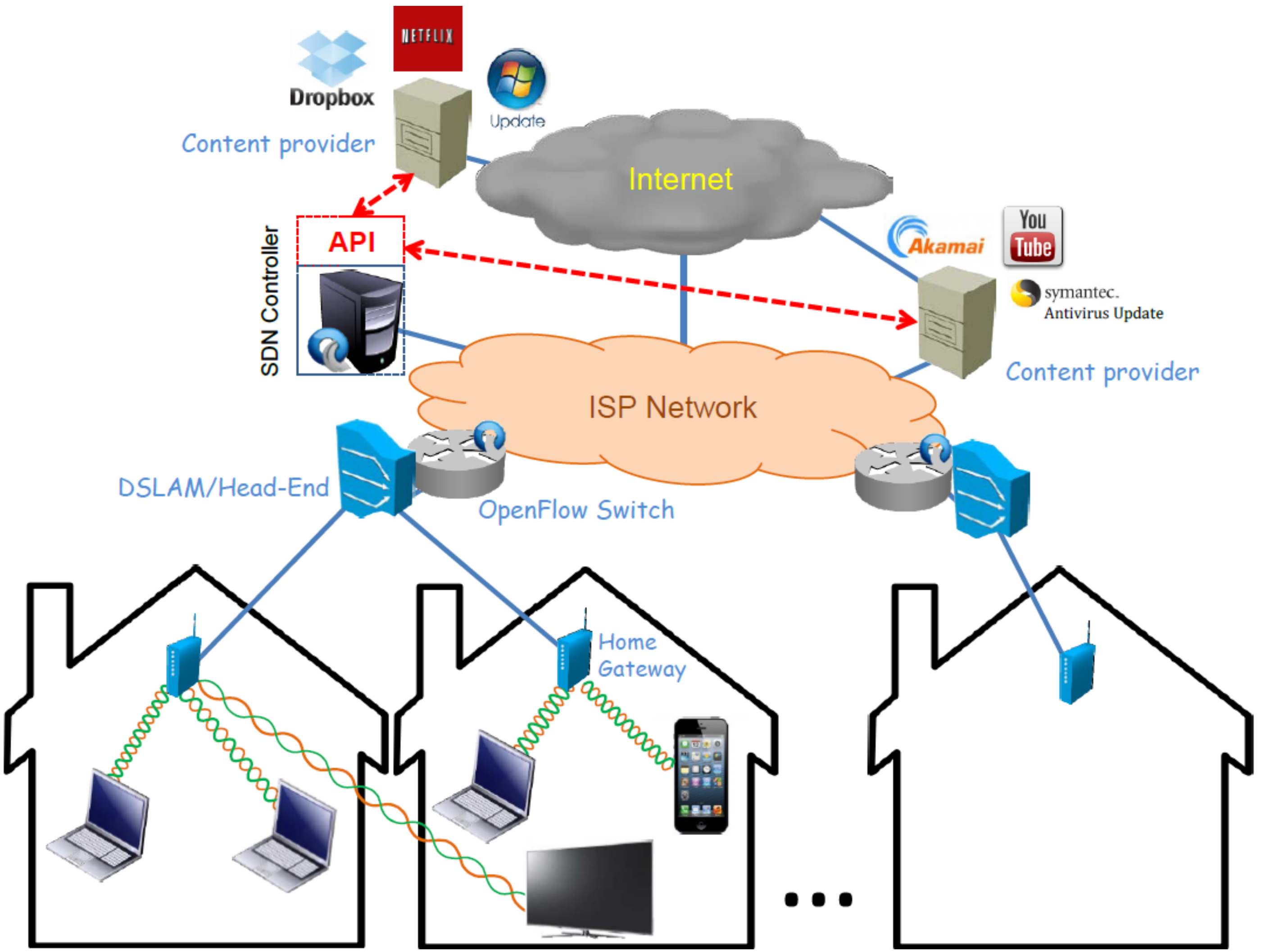
Benefits for Content Provider

- Service assurance (at a cost)
  - Consistent quality (bandwidth, jitter, loss, …)
  - Reduce application engineering effort
- Can align usage of API with business model
  - Higher QoE for premium customers
  - Tune parameters based on application/content
- Minimal changes required at content servers
  - Identify customer ISP, invoke API with the ISP
  - No changes at clients

Benefits for Users

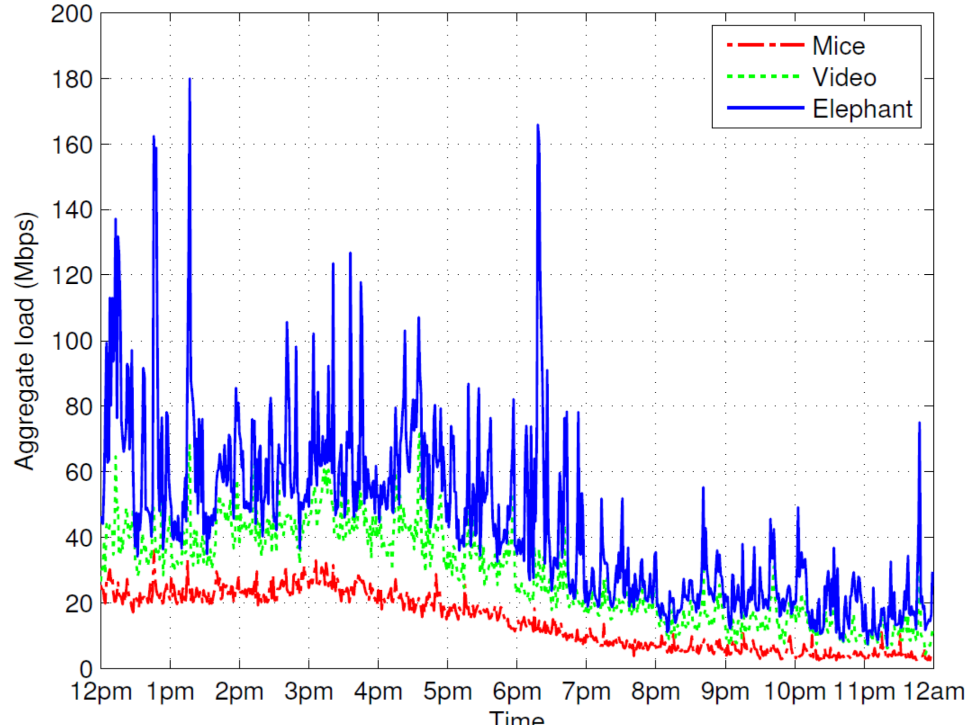
- Improved QoE
  - E.g. video bandwidth assured
- Potential for cost reduction
  - Subsidised by content provider (ads, subscriptions)
- User control and net neutrality:
  - Knob for controlling degree of virtualisation $\alpha \in [0,1]$
    - α denotes fraction of access link capacity virtualised
    - $\alpha = 0 \rightarrow$ disable; $\alpha = 1 \rightarrow$ full capacity virtualised
  - User can adjust α to suit usage/comfort

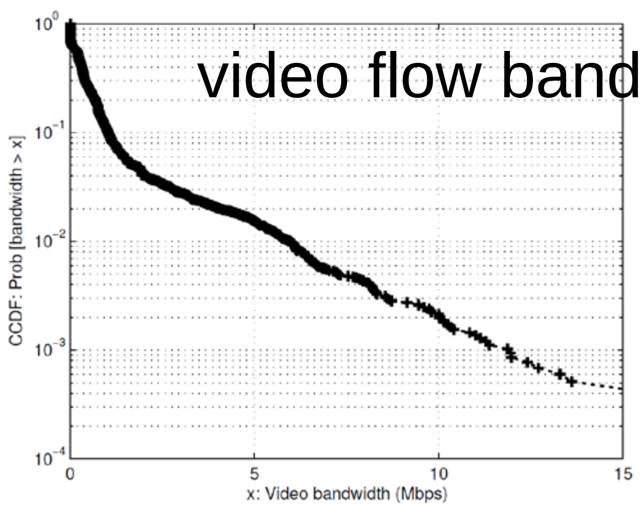
Evaluation: residential access network

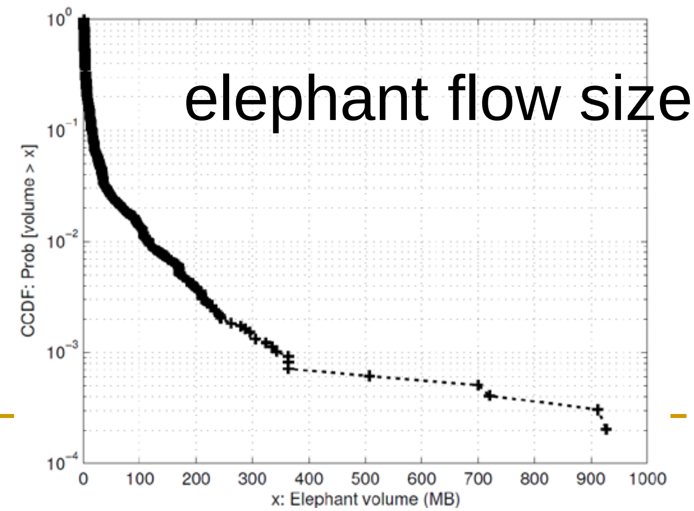


Trace Data

- UNSW campus web cache:
  - 12 hours on 16/Mar/2010
  - Flow level logs:
    - Date/time of flow arrival, Duration (mSec), Volume (Byte), Url, Content type (video, text, image)
  - 10.78 million flows, 3300 clients
- Flow categories:
  - Video (e.g. YouTube)
    - 11,674 flows
  - Mice (volume < 1MB)
    - 10.78 million flows (99.8%)
  - Elephant (volume > 1MB)
    - 9,799 flows







Simulation Setup

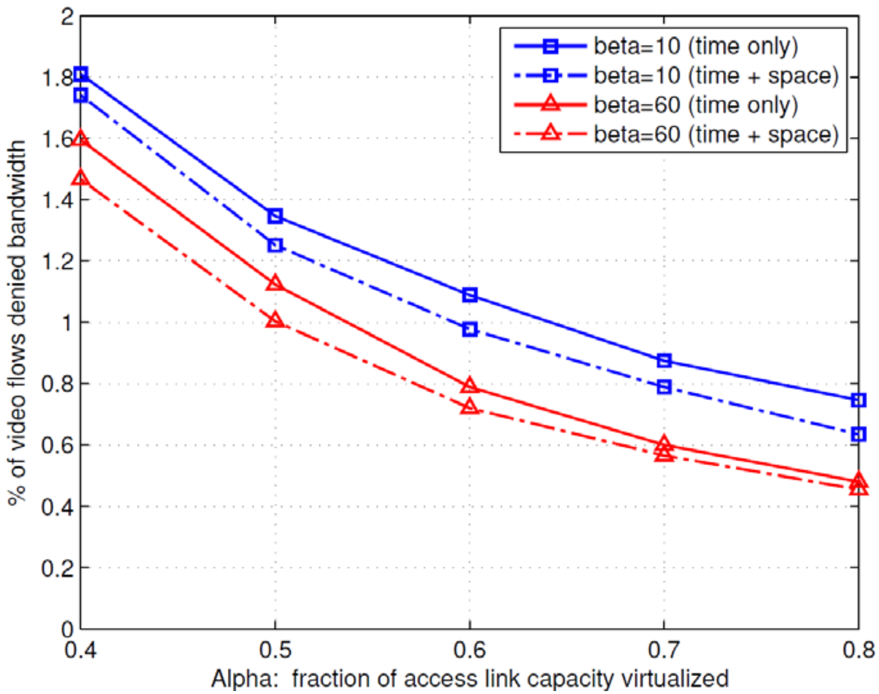
- Residential network topology:
  - 10 x four-storeyed apartment buildings
  - Each building containing 30 homes
    - Each home has a broadband capacity of 20 Mbps, and is assumed to have a wireless AP
  - WiFi overlap maps obtained for University building
    - Client within range of 5.8 APs on average
  - Clients are mapped to a randomly chosen home in a randomly chosen building
    - Roughly uniform density of 11 clients per home
- Virtualization mechanism:
  - Time scheduling (elastic traffic)
  - Space scheduling (multiple APs)

Virtualisation Algorithm

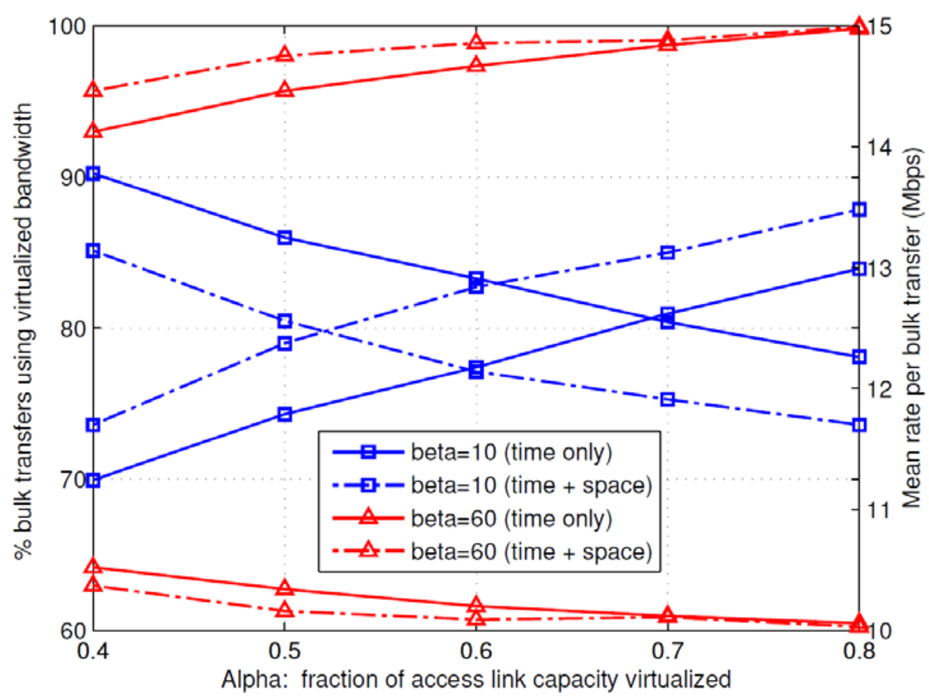
- Inputs:
  - Bandwidth requirement of (single-homed) clients
    - Video bandwidth specified in API
    - Bulk transfer bandwidth calculated periodically from deadline and size
  - Set of APs to which client can connect
- Objective: balance AP load (minimise max load)
  - Maximise chances of accepting future flows
- Output: assignment of clients to APs
- NP-hard: reduction from job shop scheduling
- Heuristic: Longest Processing Time (LPT): 4/3 OPT
  - Sort clients in descending order to bandwidth
  - Assign client to feasible AP with highest residual

Results: Allocation Failures & Bulk bw

Video allocation failures versus alpha

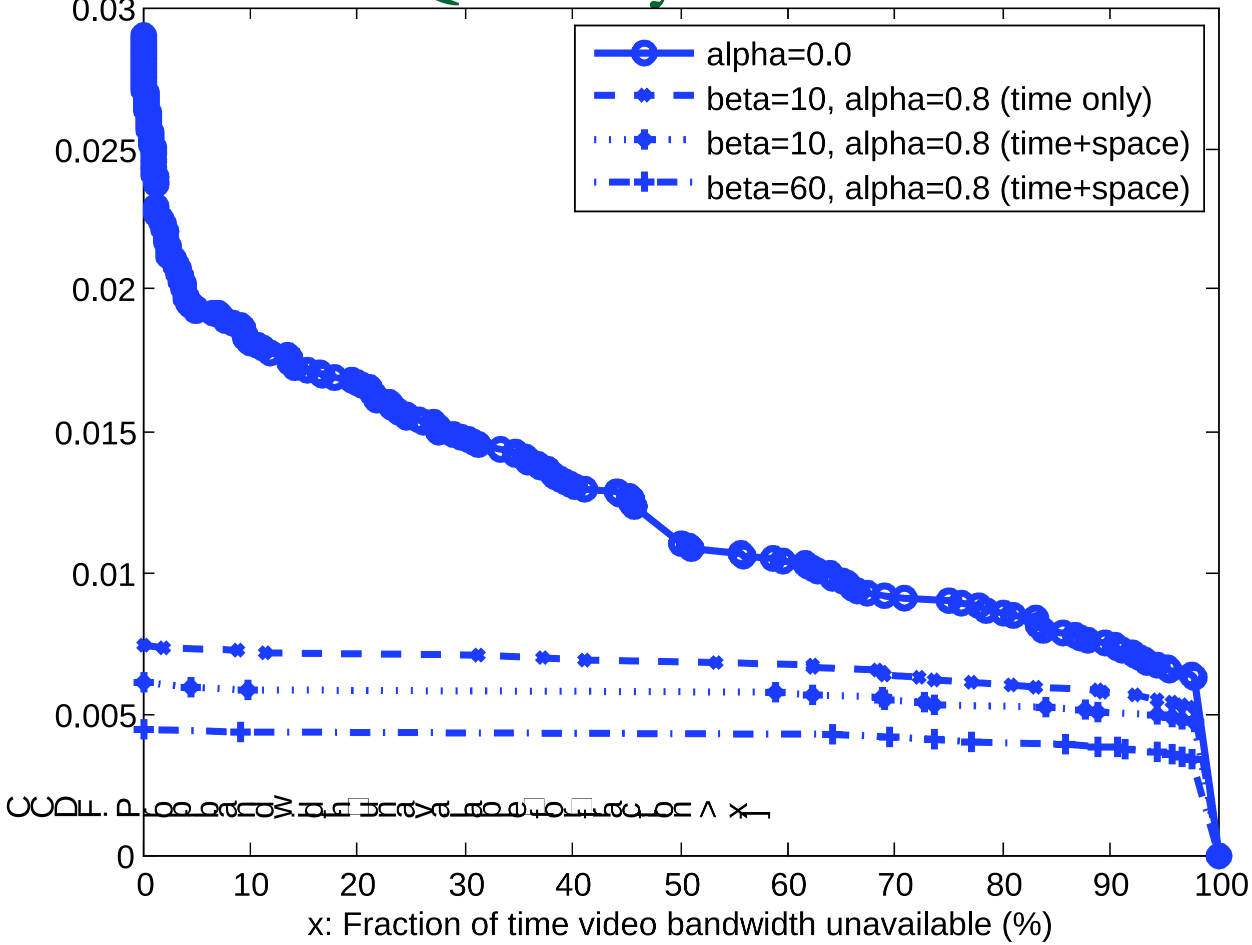


Bulk transfer allocation success and mean rate versus alpha



β = stretch factor for bulk transfers

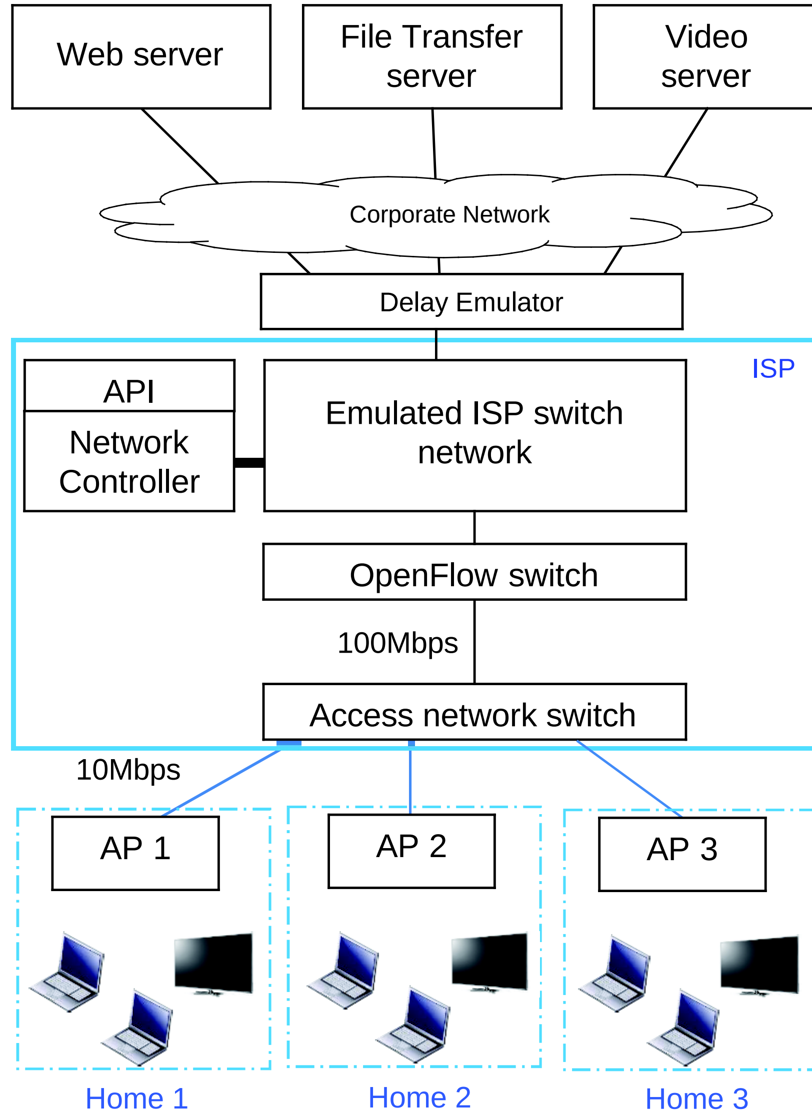
Results: Video Quality



- α = 0: about 3% flows degraded, 1% severely
- α = 0.8: about 0.8% (β=10) and 0.5% (β=60) flows degraded

UNSW THE UNIVERSITY OF NEW SOUTH WALES SYDNEY • AUSTRALIA

Test-bed @CSIRO



- Software switch OF1.0, 200 Mbps
  - Flow queue per API call, HTB slicing
- POX (python) controller
  - JSON API, runs algo periodically
- Video server: Python (Flup), VLC
- AP: TP-LINK running DD-WRTv24
- Clients: PowerShell scripted
  - C1,C2: video; C3: bulk transfer

Experimental Validation

App	$\alpha = 0$		$\alpha = 0.8$		$\alpha = 1$	
	mean	std	mean	std	mean	std
C1 MOS	2.87	0.44	3.10	0.31	3.25	0.01
C2 MOS	3.25	0.00	3.25	0.01	3.25	0.01
Page load (s)	2.84	0.86	3.10	1.61	4.85	3.55
FTP stretch	1.60	0.20	1.97	0.77	2.45	1.07

- Low-rate video (C2) always gets high MOS
- High-rate video (C1) MOS improves with α
- Web-page load-time degrades with α
- File transfers (C3) "stretch" with α

Conclusions and Future Directions

- Access network remains a bottleneck
- Motivate ISPs to “unbundle” services
  - APIs to provide per-service assurances
- End-goal: make network dynamic so it can be exposed programmatically to outside entities
- Future Work:
  - API deployment and standardisation
  - API extension to more application types
  - User-facing API and integration with home network
  - Federating API across domains