(h)ICN Socket Library for HTTP

Leveraging (h)ICN socket library for carrying HTTP messages

Mauro Sardara, Luca Muscariello, Alberto Compagno
Software Engineer

ICNRG Interim Meeting, London, 18th of March 2018

Motivations for a Socket Library

- Consistency
  - Same APIs for everyone
- Separation
  - ADU for applications PDU for Network
- Complexity
  - No layer 4 challenges for applications
- Security
  - Authentication and Integrity as built-in

Transport Services

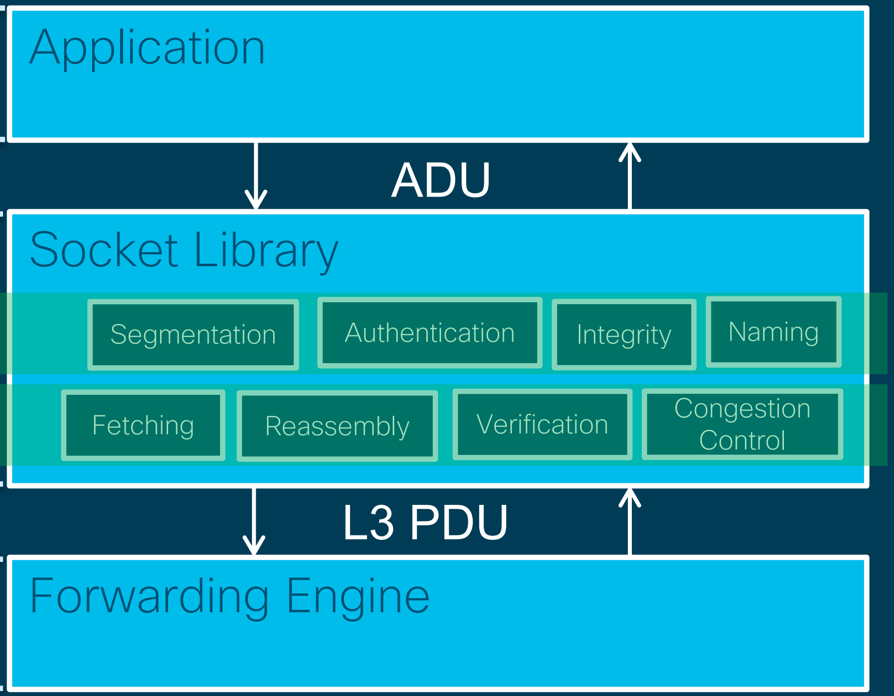
Application Logic

Transport

Producer Services

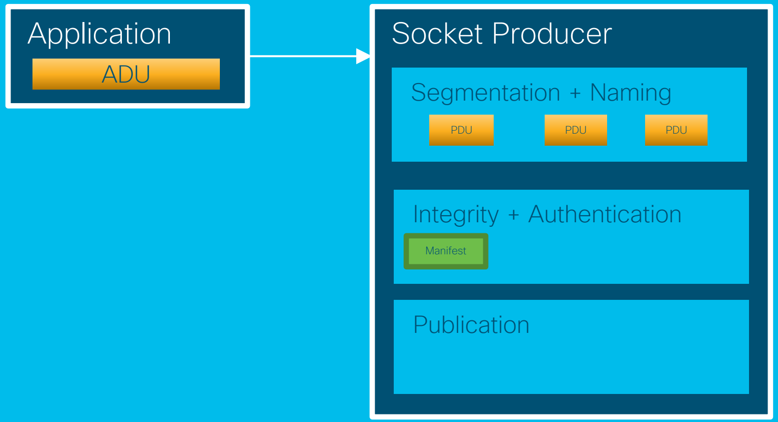
Consumer Services

Routing and Forwarding



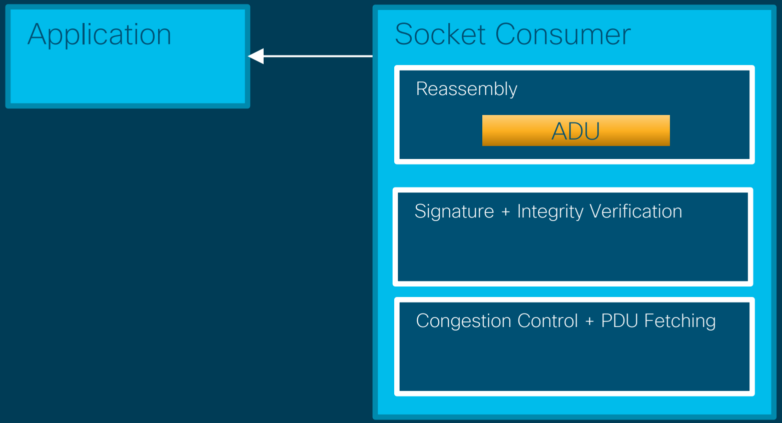
Producer Socket

- ADU Segmentation
- Naming
- Integrity
- Authentication



Consumer Socket

- Congestion Control
- PDU Fetching
- Signature and Integrity verification
- ADU Reassembly



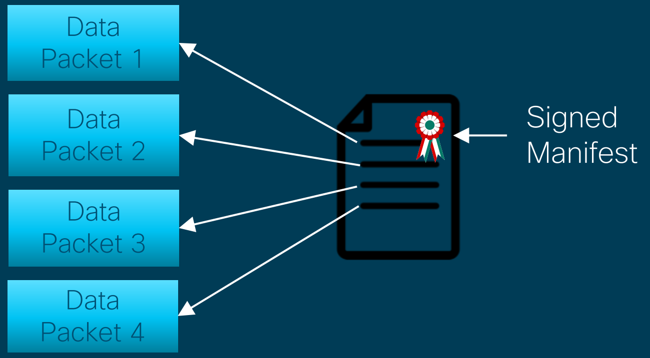
Transport Manifest

- Metadata and prefetching
  - Network names of next data to pull
- Signature Verification
  - Manifest always signed
- Integrity Verification
  - It contains hashes of contents that are going to be pulled
- Performance
  - Amortizes verification cost of each content object

Authentication and Integrity

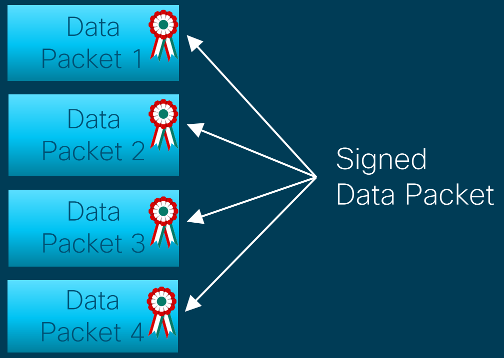
- Native security features, transparently offered to each application
- 2 Approaches: **Manifest authentication** vs **per packet authentication**

- Manifest Authentication



Integrity verified with HASH inside Signed Manifest.

- Per Packet Authentication



Integrity verified with the signature itself

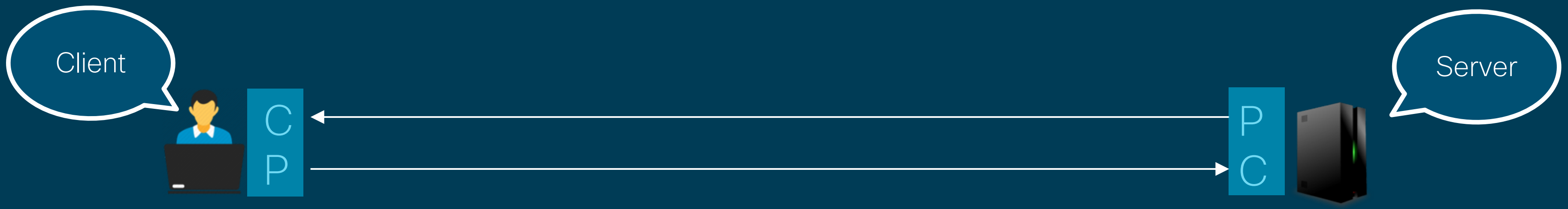
fd.io open source project

```
git clone -b libicnet/master https://gerrit.fd.io/r/cicn libicnet;
```

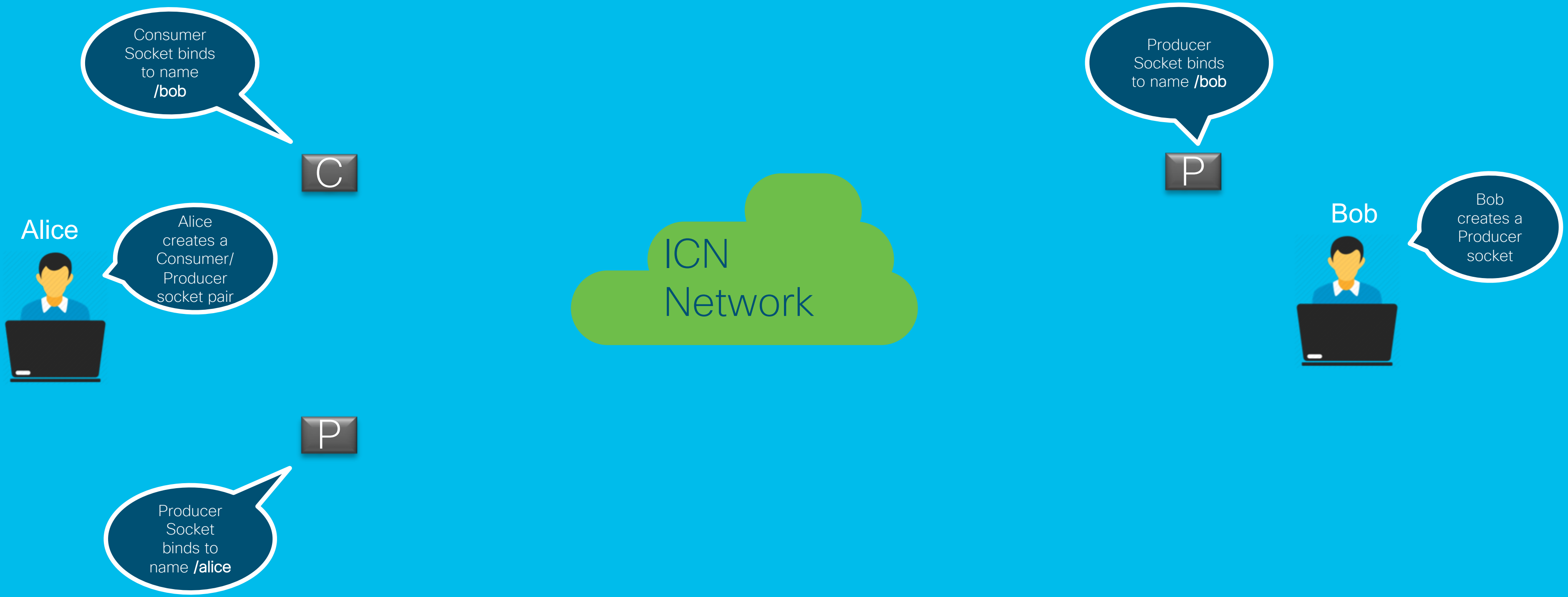
- The ICN transport library, called Libicnet, has been open sourced under APACHE 2.0 license on FD.io (https://wiki.fd.io/view/Cicn)
- Language: C++
- Supported Platforms: Ubuntu, CentOS, macOS, iOS, Android
- It allows applications to transparently connect to 2 ICN Forwarders:
  - sb-forwarder (a.k.a. Metis)
  - cicn-forwarder (vpp based plugin)
- Applications written using this library are able to work also with the hICN library (libhicnet)

Bidirectional data exchange between two applications

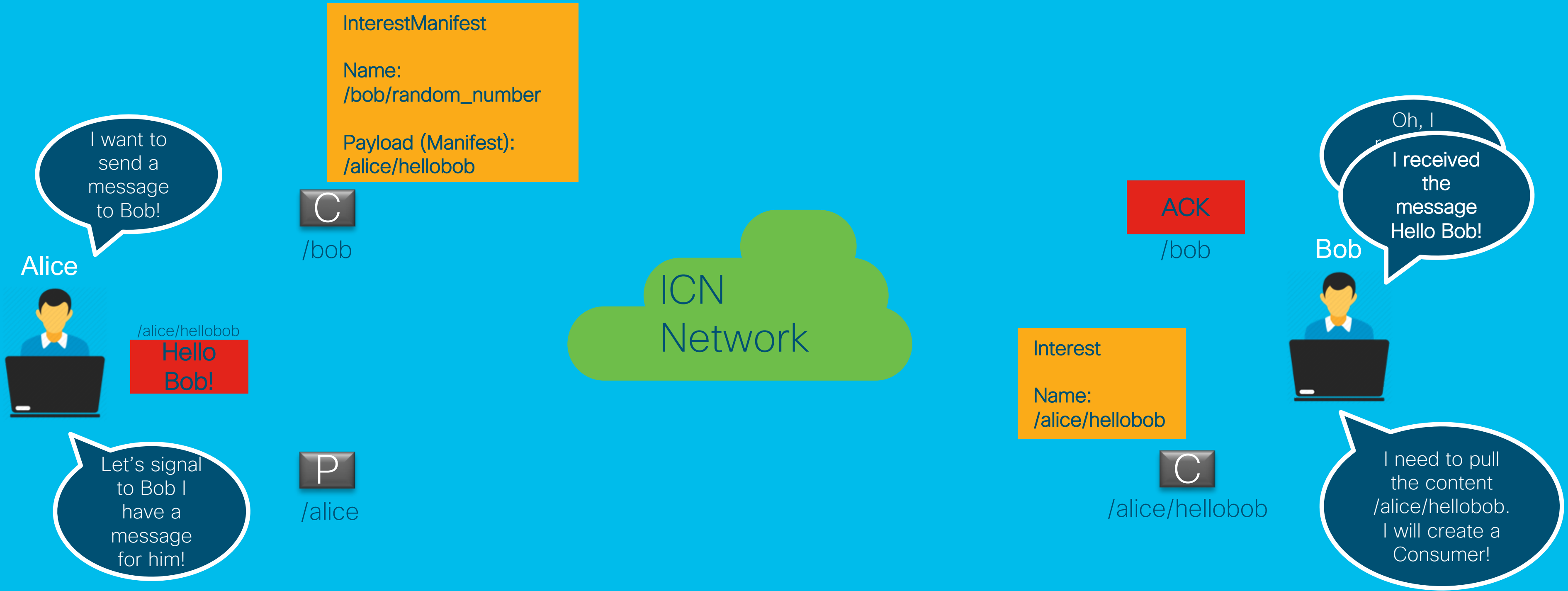
- Each socket identifies an unidirectional channel (multi-point)
- By using Producer/Consumer sockets an application is able to Send/Receive data
  - ➢ The Consumer Socket pulls contents from a Producer socket
  - ➢ The Producer Socket publishes data to be pulled by a Consumer Socket



Example: Implementing push semantics using the reverse pull(1)



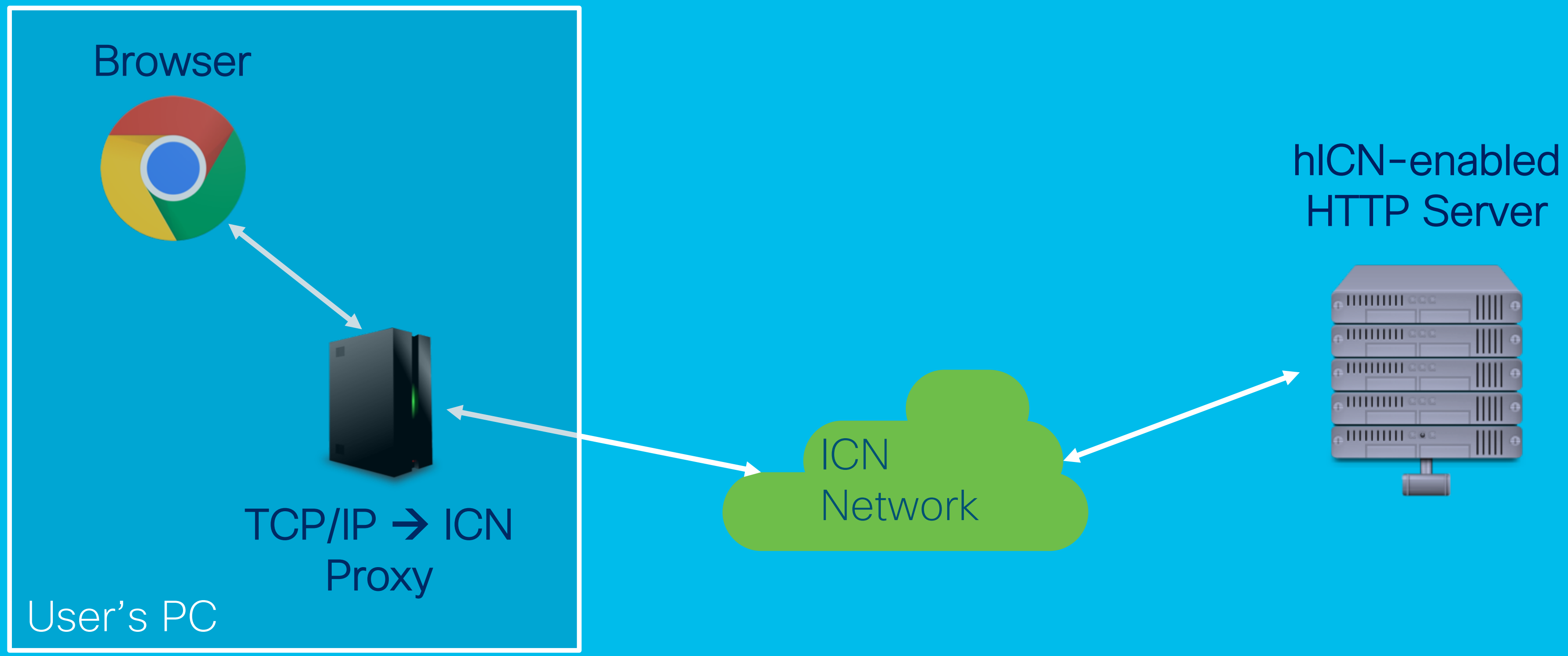
Example: Implementing push semantics using the reverse pull(2)



HTTP Client and Server

- The concept explained in the previous slides can be easily applied for achieving full HTTP Client/Server communications.
- Clients send HTTP Requests in the same way as explained before
- Servers process Requests and publish Responses
- Clients pull back Responses

HTTP over ICN PoC



Optimizations

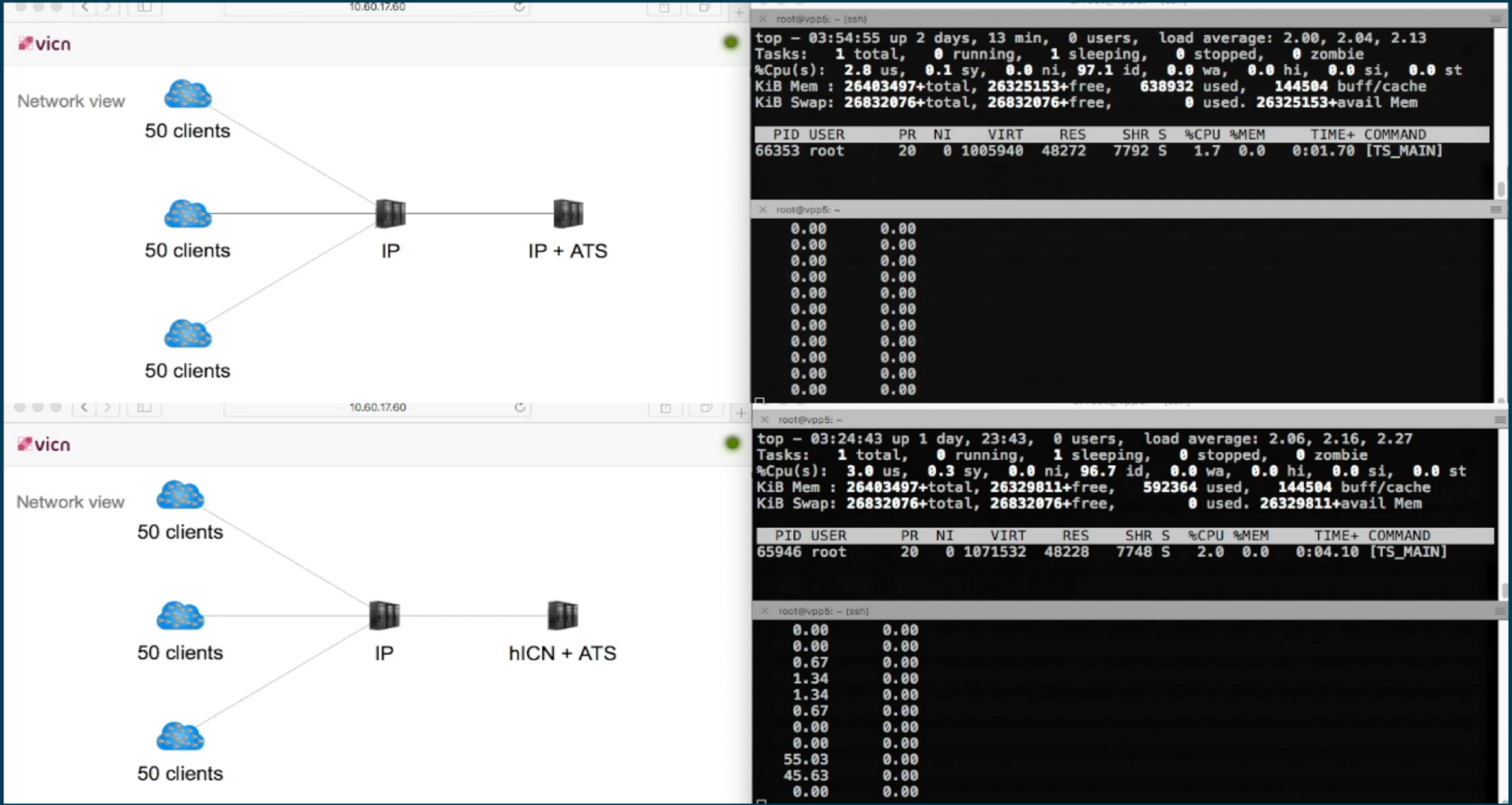
- If the size of the HTTP requests fits one MTU, it can be directly piggybacked within the first interest manifest, by sending the request in half RTT.
  - The SPDY whitepaper states that typically Request header size of 700-800 bytes is common (https://www.chromium.org/spdy/spdy-whitepaper)
- When the client sends the interest manifest to the server, the latter can append to the ACK a signed manifest containing the information for retrieving the response, allowing clients to directly retrieve it.

HTTP Request and Response multiplexing

- It refers to the problem of sending multiple requests/responses in parallel
- The client must be able to associate each response to the corresponding request
  - HTTP 1.0 uses multiple TCP connections for multiplexing Requests and Responses
  - HTTP 1.1 can reuse the same TCP connection, but for being able to associate Responses and Requests the server needs to process them in order, likely causing a HOL blocking
  - HTTP 2.0 uses one persistent TCP connections and streams, with the overhead of demultiplexing at application layer
- ICN solves this problem by associating to each request/response a different name prefix: the client always knows what request originated a certain response, so it can easily send multiple requests in parallel.

Scalability PoC: Multicast and Server Load

- This experiment shows relevant benefits in using HTTP with an ICN transport, in particular for scalability at Server Side
- We consider the case of DASH **linear video distribution**:
  - Cluster of 150 clients connected to an ICN enabled Apache Traffic Server (ATS)
    - Reverse Proxy, 2 GB cache, nginx origin server serving 48 channels
  - Each client requests one of the 48 available channels (*zipf* distribution, $\alpha=1.4$)
  - An HTTP Request can be directly served by the transport (rather than by ATS), if the corresponding Response has been already published in the Producer socket output buffer
- The video distribution <u>scales with the number of active channels</u> instead of the number of active users as using a TCP/IP network.
  - Server load (Memory/CPU) considerably reduced with ICN transport


10.60.17.60
vicn
Network view
50 clients
50 clients
50 clients
IP
IP + ATS
root@vpp5: ~ (ssh)
top - 03:54:55 up 2 days, 13 min, 0 users, load average: 2.00, 2.04, 2.13
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.8 us, 0.1 sy, 0.0 ni, 97.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 26403497+total, 26325153+free, 638932 used, 144504 buff/cache
KiB Swap: 26832076+total, 26832076+free, 0 used. 26325153+avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
66353 root 20 0 1005940 48272 7792 S 1.7 0.0 0:01.70 [TS_MAIN]
root@vpp5: ~
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
10.60.17.60
vicn
Network view
50 clients
50 clients
50 clients
IP
hICN + ATS
root@vpp5: ~
top - 03:24:43 up 1 day, 23:43, 0 users, load average: 2.06, 2.16, 2.27
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.0 us, 0.3 sy, 0.0 ni, 96.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 26403497+total, 26329811+free, 592364 used, 144504 buff/cache
KiB Swap: 26832076+total, 26832076+free, 0 used. 26329811+avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
65946 root 20 0 1071532 48228 7748 S 2.0 0.0 0:04.10 [TS_MAIN]
root@vpp5: ~ (ssh)
0.00 0.00
0.00 0.00
0.67 0.00
1.34 0.00
1.34 0.00
0.67 0.00
0.00 0.00
0.00 0.00
55.03 0.00
45.63 0.00
0.00 0.00

cisco