A Transport Layer and Socket API for (h)ICN

Design, Implementation and Performance Analysis

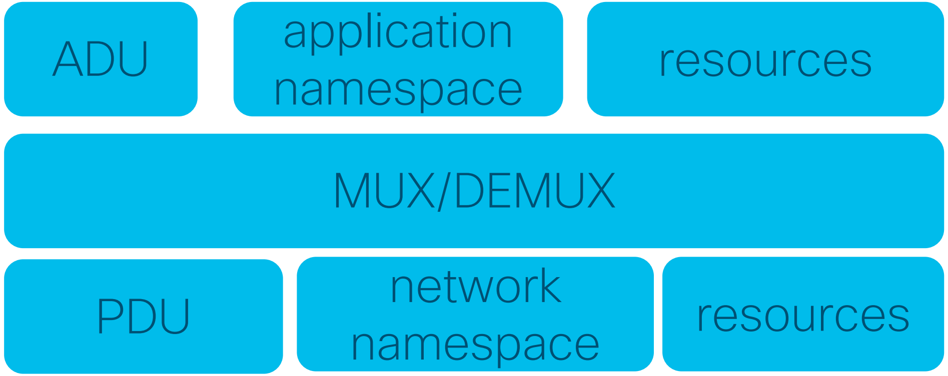
Luca Muscariello, Mauro Sardara, Alberto Compagno

ICN RG Interim Meeting – Montreal – 15th of July 2018

Motivation

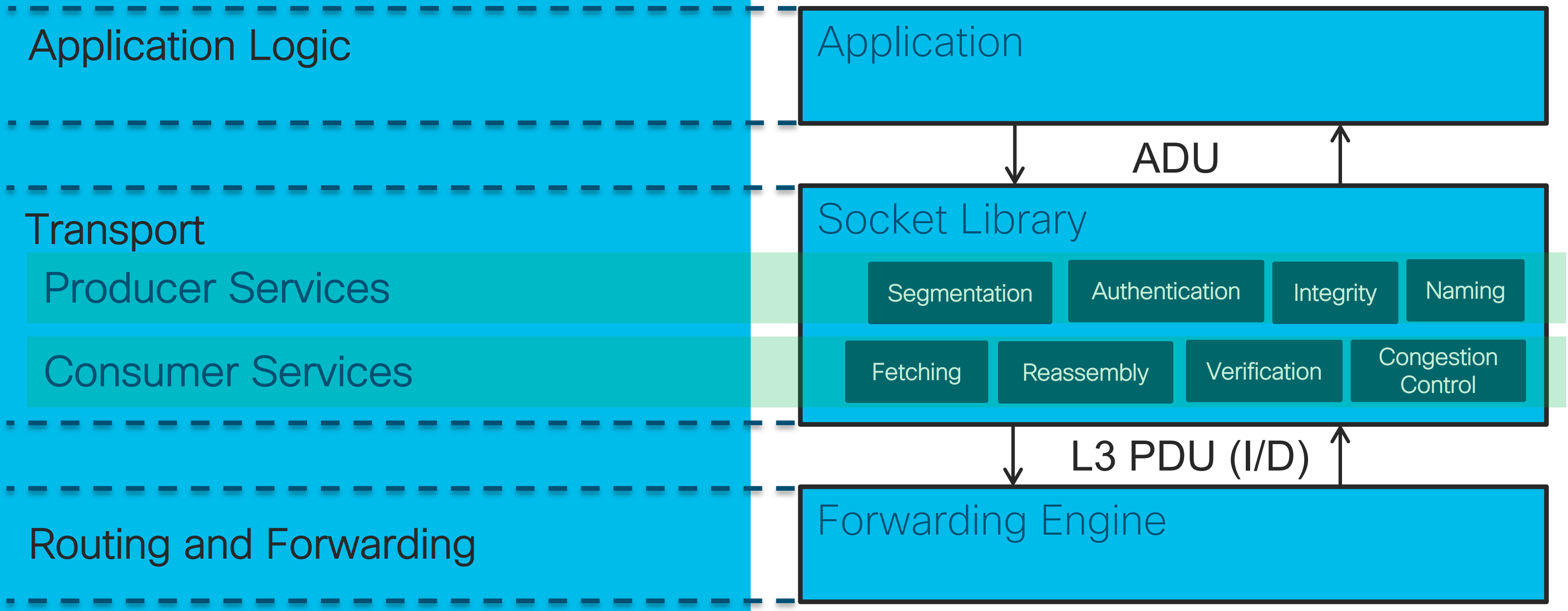
- provide new API for apps
- socket API
- post-socket API
- evolutionary deployment of (h)ICN in existing applications
- portability
- (h)ICN = {NDN, CCN, hICN}

Protocol layers and namespaces

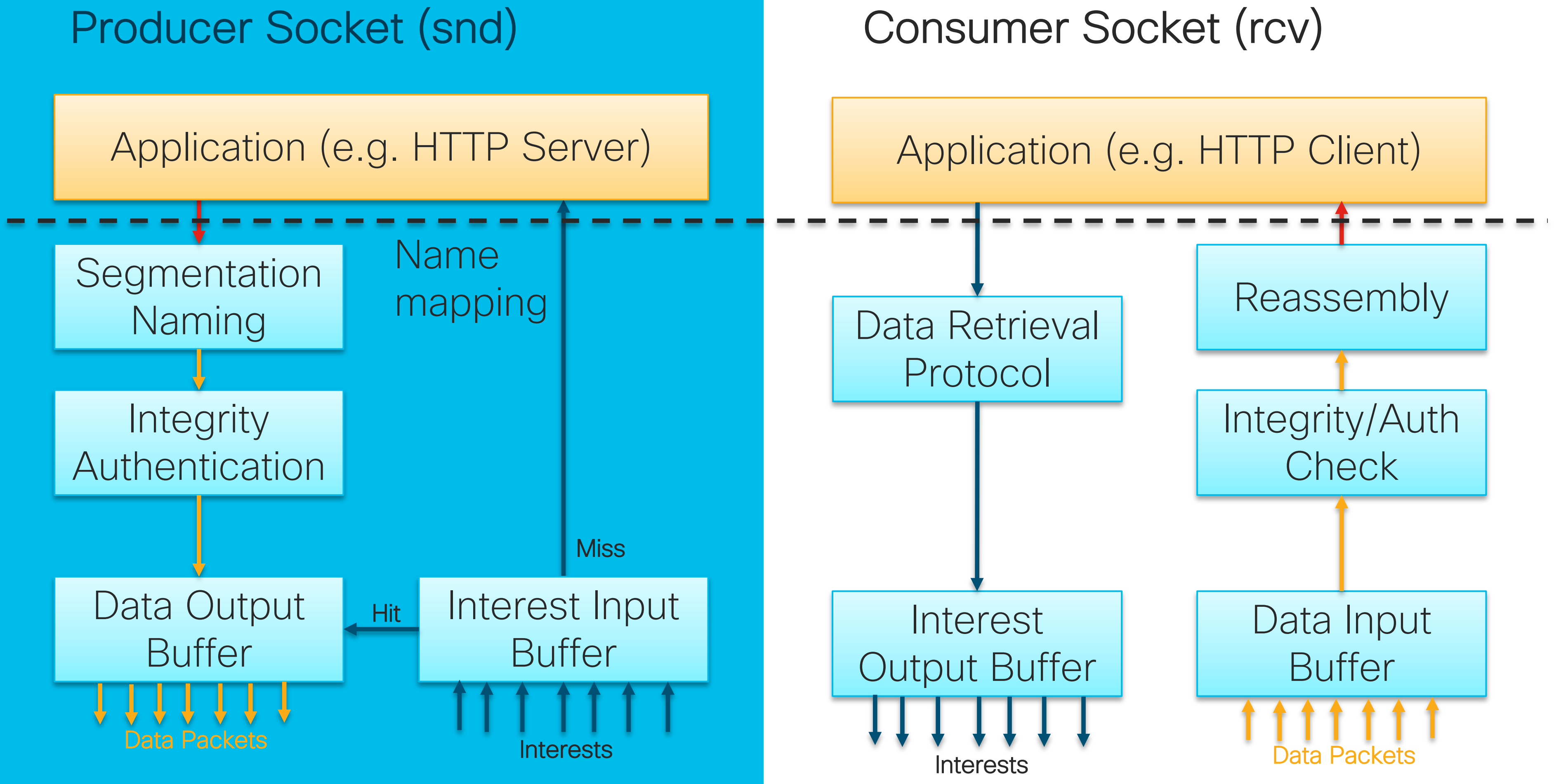


- Application namespaces
  - Manage application resources
  - Longer lifetime
  - Scope and delegation can be local or global to the participants
- Network namespaces
  - Used to manage network resource sharing
  - Shorter lifetime
  - network resources are highly constrained
- Resources are multiplexed and demuliplexed by solving a resource allocation problem

Transport Services in (h)ICN



unidirectional channels



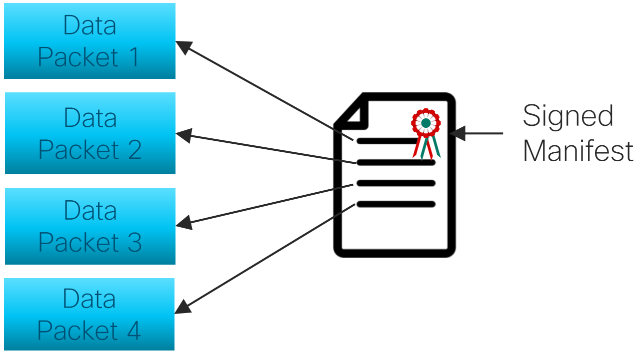
Transport Manifest

- Indexing
  - network names mapping and synchronization between consumer/producers sockets
- Signature Verification
  - Manifest always signed
- Integrity Verification
  - It contains hashes of data packets
- Performance
  - Amortizes verification cost of each content object

Authentication and Integrity

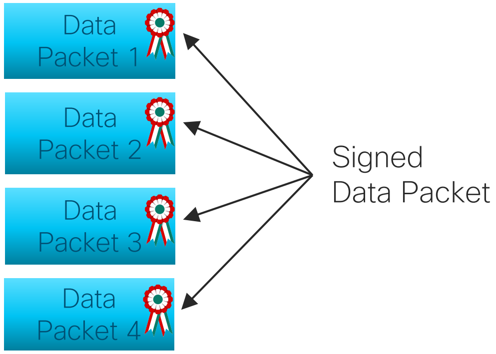
- Native security features, transparently offered to each application
- Two approaches: **Manifest authentication** vs **per packet authentication**

- Manifest Authentication



Integrity verified with hash inside the signed manifest.

- Per Packet Authentication



Integrity verified with the signature itself

Implementation

- Transport stack implemented as C++ user-space library
- Client and Server stack
- Server stack based on VPP and DPDK
- Client stack as a portable app
- Open sourced under APACHE 2.0 license on FD.io (https://wiki.fd.io/view/Cicn)
- Supported Platforms: Linux, macOS, Android, iOS, Window

(h)ICN Transport Services API

- How transport services are exposed to applications?
  - Legacy BSD socket API
    - socket(), send(), recv(), bind()…
    - Simple, clear and widely adopted…
    - …yet limited and difficult to integrate in an evolving Internet
  - Modern post-socket API
    - Intent based, event driven asynchronous interface
    - Simple and more expressive than BSD sockets
    - Easy to integrate with the variety of internet protocols and access technologies

(h)ICN with BSD socket API

- Take hICN namespaces based on IPv6 prefixes
- Based on socket interface extension for IPv6[1]
- Domain: `AF_ICN/AF_HICN address and protocol family`
- Socket types: `SOCK_CONS` and `SOCK_PROD`
- Protocol types: `CONS_REL/CONS_UNREL` and `PROD_REL/PROD_UNREL`

Example: publish a content using a SOCK_PROD

```
// Create a new producer socket
int sockfd = hicn_socket (AF_HICN, SOCK_PROD, PROD_REL);

// Set the namespace for publishing contents
struct sockaddr_hicn producer_namespace {
  .sin_family = AF_HICN,
  .sin_prefix = 64 // Limit the namespace to a /64 namespace
};

hicn_pton(AF_HICN, "b001:a:b:c::", &producer_namespace. sin_addr);

// Bind a local face with forwarder and set the corresponding local route
hicn_bind(sockfd, &producer_namespace, sizeof(producer_namespace));

// Publish a content with name b001:a:b:c::1234
struct sockaddr_hicn content_name {
  .sin_family = AF_HICN,
};
hicn_pton(AF_HICN, "b001:a:b:c::abcd", &producer_namespace.sin_addr);

char *buf = "This string will be published in the hicn socket";
ssize_t n = hicn_sendto(sockfd, buf, BUF_SIZE, 0, &content_name, NULL);
```

Example: pull a content using a SOCK_CONS

```c
// Create a new consumer socket
int sockfd = hicn_socket (AF_HICN, SOCK_CONS, CONS_REL);

// Set the name of the content to pull
struct sockaddr_hicn content_name {
  .sin_family = AF_HICN,
  .sin_prefix = 64 // Limit the interest namespace to a /64 namespace
};

hicn_pton(AF_HICN, "b001:a:b:c::1234", &content_name.sin_addr);

// Bind a local face with forwarder and enforce the namespace size limit
hicn_bind(sockfd, &content_name, sizeof(content_name));

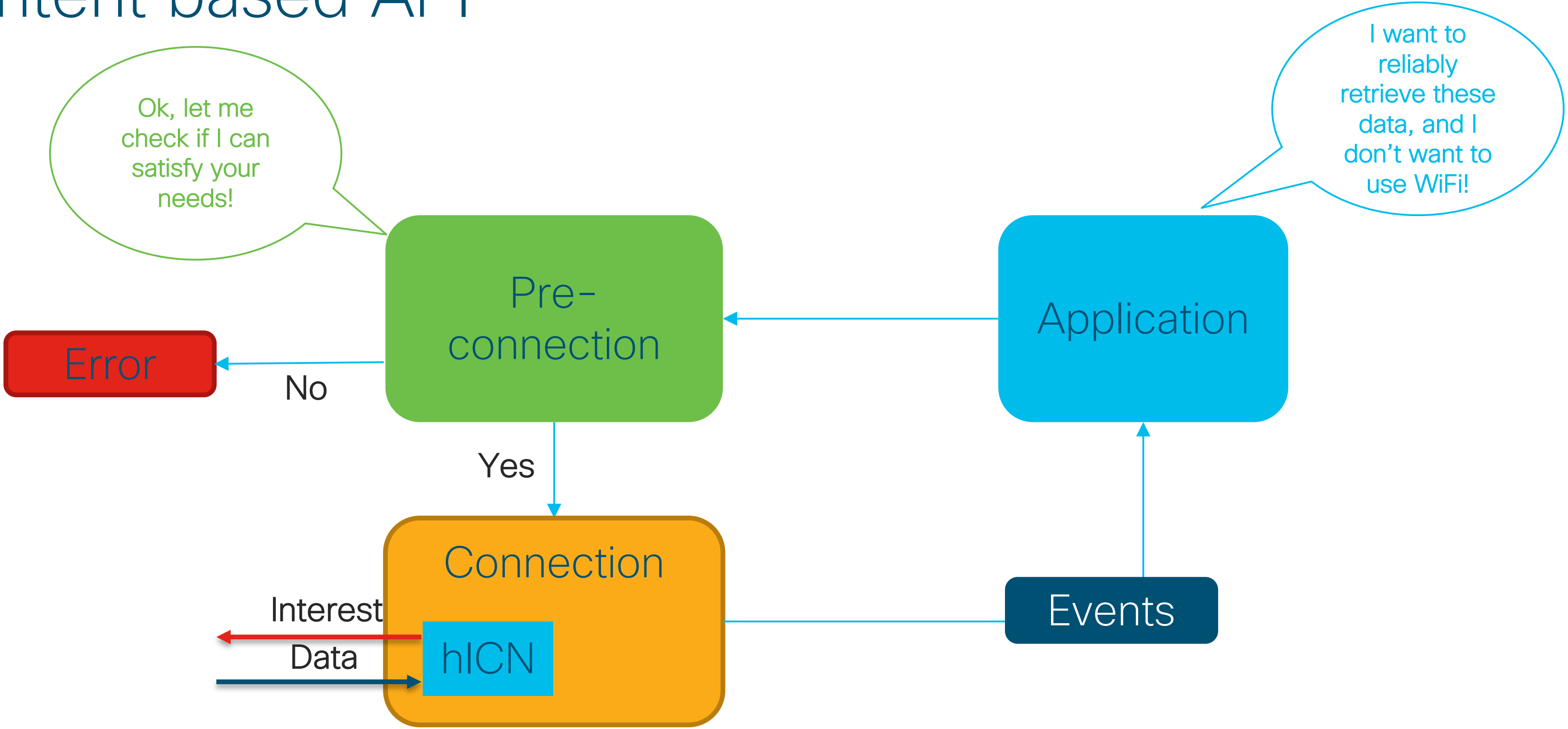
// Pull the content with name b001:a:b:c::1234 and store it inside buf
char buf[BUF_SIZE];
ssize_t n = hicn_recvfrom(sockfd, buf, BUF_SIZE, 0, &content_name, NULL);
```

hICN with Post-Socket API[1]

- Generic API: application describes the required transport service, enforcing preferences and constraints
- hICN is a transport service provided by the system
- If application requirements can be satisfied with an hICN transport protocol, it is selected and used

[1] Tommy Pauly, Brian Trammell, Anna Brunstrom, Gorry Fairhurst, Colin Perkins, Philipp S. Tiesel, and Christopher A. Wood. 2018. An Architecture for Transport Services. Internet-Draft draft-pauly-taps- arch-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-pauly-taps-arch-00

Intent based API



Interface objects

- Pre-connection
  - Set of parameters and constraints of the application on path/protocol selection
  - It is used for electing a transport service able to satisfy application needs
- Connection
  - Transport protocol stack on which data can be received
  - Send interests to a remote endpoint (data name)
- Listener
  - Listen for incoming interests and publishes application data
  - Set up of a local endpoint (production namespace)

Example: publish data using post-socket APIs

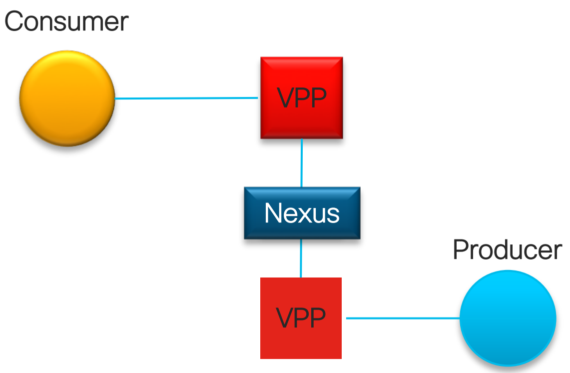
```
class HIcnProducer {
...
void onConnectionReceived(Error &error, Connection &connection, Name &name) {
  // Publish data upon consumer request reception
}
void run() {
  // Describe the desired transport
  transport_parameters_.setPreference(TransportProperty::TRANSPORT_MANIFEST, PreferenceLevel::REQUIRE);
  transport_parameters_.setPreference(InterfaceType::ETHERNET, PreferenceLevel::PROHIBIT);
  security_parameters_.add(SecurityProperty::IDENTITY, producer_identity_);
  security_parameters_.setPreference(SecurityProperty::SIGNATURE, PreferenceLevel::REQUIRE);
  // Describe the endpoint
  local_endpoint_.add(EndpointProperty::NAMESPACE, "b001:a:b:c::/64");
  // Use Wifi
  local_endpoint_.add(EndpointProperty::INTERFACE, Interfaces::LTE);
  // Create preconnection
  preconnection_ = new Preconnection(remote_endpoint_, transport_parameters_, security_parameters_);
  // Initiate a new listener starting from the preconnection.
  listener_ = preconnection_->listen();
  // Packetize a buffer of data, using manifests
  listener_->publish("b001:a:b:c::1234", buffer, size);
  // Wait for events on the socket
  listener_->processEvents(onConnectionReceived);
}
private:
 LocalEndpoint local_endpoint_;
 TransportParameters transport_parameters_;
 SecurityParameters security_parameters_;
 Identity producer_identity_;
 Preconnection *preconnection_;
 Listener *listener_;
};
```

Example: pull a content using post-socket API

```
class HIcnConsumer {
...
void onContentReceived(Error &error, Connection &connection, Name &name) {
  // Process received data..
}
void run() {
  // Describe the desired transport
  transport_parameters_.setPreference(TransportProperty::TRANSPORT_MANIFEST, PreferenceLevel::REQUIRE);
  transport_parameters_.setPreference(TransportProperty::RELIABLE_DATA_TRANSFER, PreferenceLevel::REQUIRE);
  transport_parameters_.setPreference(TransportProperty::DATA_ORIGIN_VERIFICATION, PreferenceLevel::REQUIRE);

  // Describe the endpoint
  remote_endpoint_.add(EndpointProperty::NAME, "b001:a:b:c::1234");
  // Use LTE
  remote_endpoint_.add(EndpointProperty::INTERFACE, Interfaces::LTE);
  // Create preconnection
  preconnection_ = new Preconnection(remote_endpoint_, transport_parameters_, security_parameters_);
  // Initiate a new connection starting from the preconnection.
  connection_ = preconnection_->initiate();
  // Asynchronously start the content download
  connection_->asyncRecv(buffer, size, onContentReceived);
  // Wait for events on the socket
  connection_->processEvents();
}
private:
 RemoteEndpoint remote_endpoint_;
 TransportParameters transport_parameters_;
 SecurityParameters security_parameters_;
 std::string producer_public_key_;
 Preconnection *preconnection_;
 Connection *connection_;
};
```

Performance Evaluation of the server stack



- segmentation and reassembly
- crypto operations
- no hardware offloading involved

- Cisco UCS Type-C
- Intel Xeon CPU E5-2695 v4
- 45 MB cache, 2,10 GHz, 18 cores
- 256 GB of RAM
- Intel 82599ES 10-Gbps NIC
- 10Gbps Cisco-Nexus 5k

- Best performance obtained by
- NUMA nodes aware configuration of the hardware and software.
- provide CPU affinity, reduce context switching to minimum
- avoid PCI bus bottleneck

Performance Evaluation

- TCP vs hICN REL transport service
- TCP newreno for VPP stack
- TCP Cubic for Linux stack

- hICN REL_CONS transport service w/ options:
  - Window based flow control (AIMD)
  - delay-based congestion control
  - w/ or w/o manifest
  - w/ or w/o ADU prefetching

- hash is SHA-256
- Signatures RSA-1024 or ECDSA-192

Type of test	Average	99% CI
(h)ICN Asynchronous Publication		
Manifest RSA-1024	928Mbps	[919 936]
Packet-wise RSA-1024	290Mbps	[283 297]
Manifest ECDSA-192	531Mbps	[523 538]
Packet-wise ECDSA-192	28Mbps	[27 28]
(h)ICN Synchronous Publication		
Manifest RSA-1024	525Mbps	[518 532]
Packet-wise RSA-1024	26Mbps	[26 27]
Manifest ECDSA-192	530Mbps	[522 537]
Packet-wise ECDSA-192	28Mbps	[28 29]
(h)ICN Crypto Operations disabled		
No signature	5.79Gbps	[5.77 5.81]
No signature, 2 transfers	5.80Gbps	[5.73 5.86] Jain=0.99
No signature, 3 transfers	6.46Gbps	[6.44 6.49] Jain=0.98
TCP - Iperf		
Linux TCP (w/ TSO)	9.19Gbps	[9.09 9.30]
Linux TCP (w/o TSO)	5.00Gbps	[4.88 5.12]
VPP TCP stack	9.24Gbps	[9.22 9.26]

Hardware offloading and software optimization

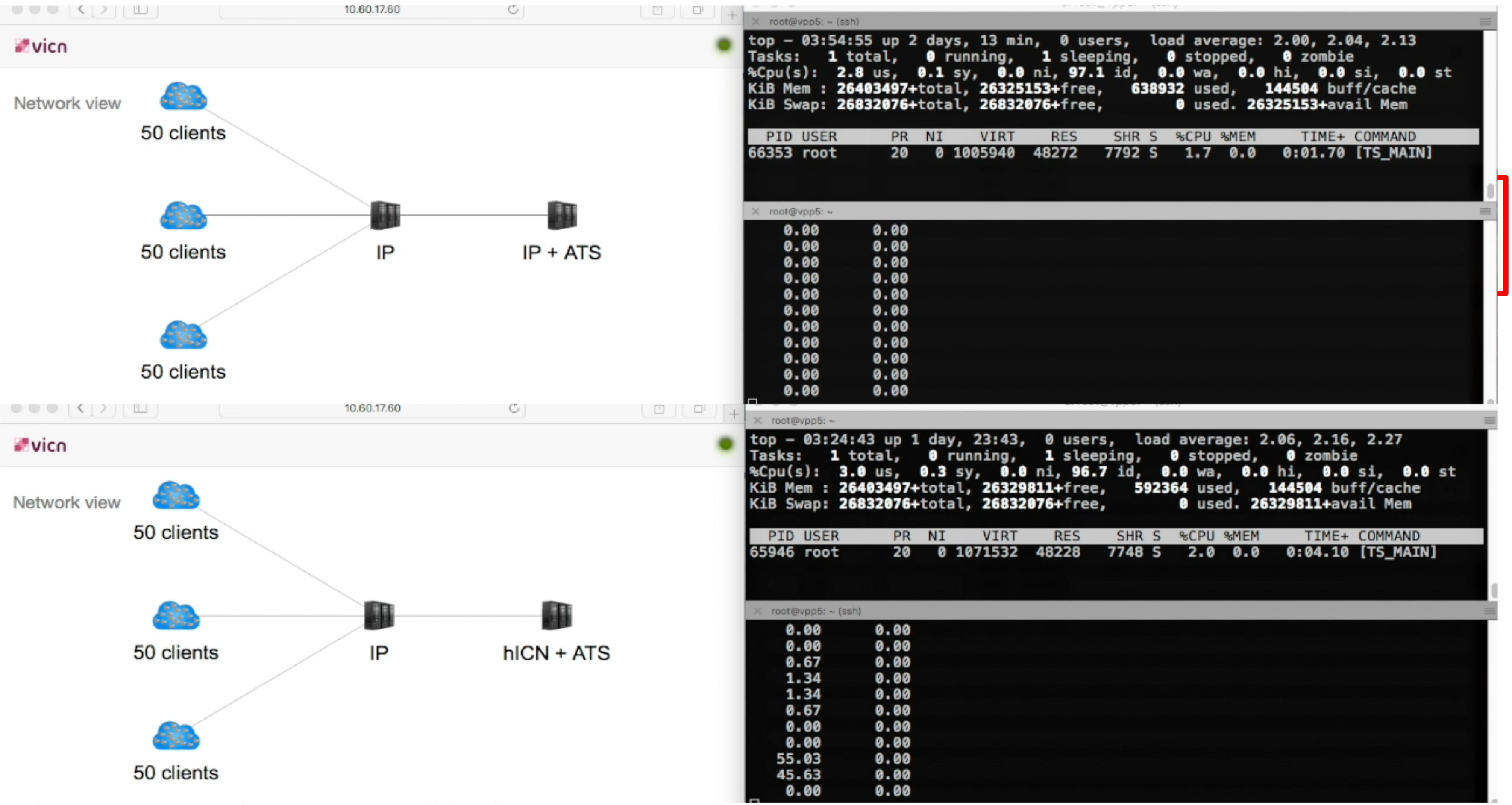
- HW offloading
- Crypto
- Intel OAT (I/O acc. tech)
- arm acceleration
- Generic Segmentation Offload
- Generic Receive Offload
- Portability and features support tradeoffs.

- SW optimization such as VPP
- Based on batching
- Reduce memory access
- Improve core optimization
- to be used carefully for latency/goodput tradeoffs

Conclusions

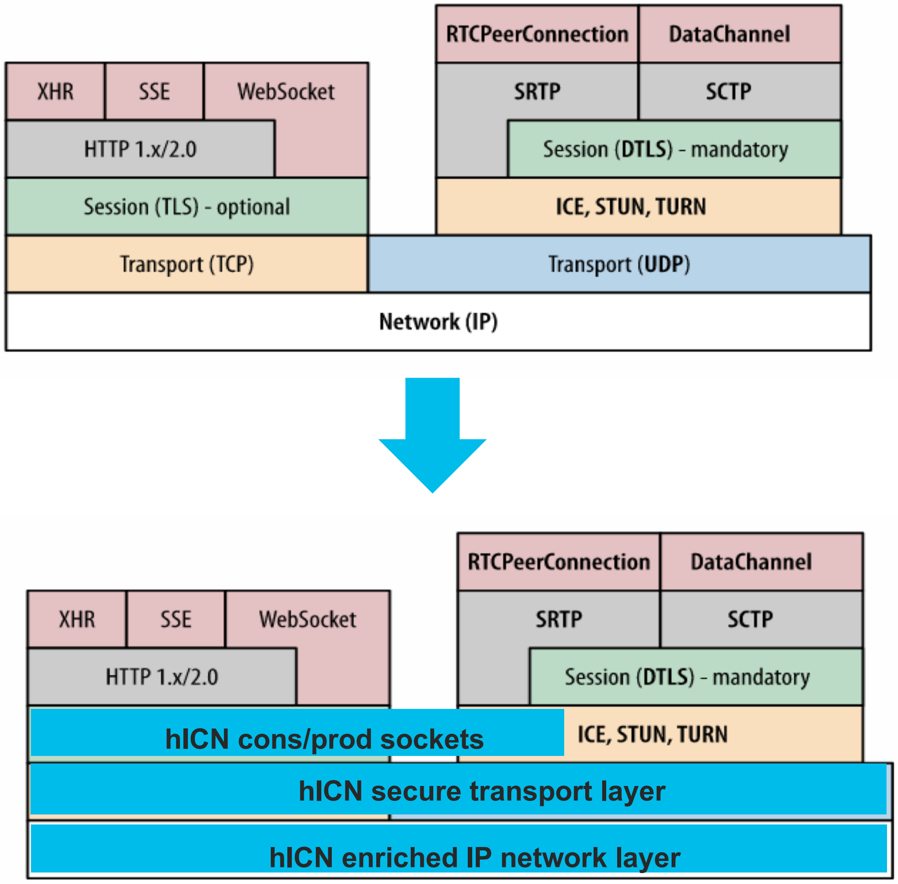
- The development of API new is crucial to facilitate integration
- Backward compatible integration for existing applications
- Advanced applications such as AR/VR, IoT can benefit from novel API and transport services
- Deployment in software, user-space integration
- High-speed for server stack
- Highly portable for client stack
- Hardware offloading is an important topic to be supported as much as possible

Demos


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

WebRTC over hICN

- New sockets (reuse of TCP or UDP header)
- Mapping of application to network names(RTP and HTTP)
- Replacement of transport with hICN transport for some or all communications
- hICN-enriched IP network layer (names, forwarding strategies)



Backup slides

hICN with BSD socket API

- Common API (Consumer/Producer)
  - `int hicn_socket (int domain, int socket_type, int protocol)`
  - `int hicn_bind(int sockfd, struct sockaddr *addr, socklen_t addrlen)`
  - `ssize_t hicn_sendmsg(int sockfd, const struct msghdr *msg, int flags)`
  - `ssize_t hicn_recvmsg(int sockfd, struct msghdr *msg, int flags)`
  - ```
    int hicn_setsockopt (int sockfd, int level, int __optname, const void *optval,
                         socklen_t optlen);
    ```
- Consumer specific API
  - ```
    ssize_t hicn_recvfrom (int sockfd, void *buf, size_t n, int flags,
                           struct sockaddr * addr, socklen_t *addr_len);
    ```
- Producer specific API
  - ```
    ssize_t hicn_sendto (int sockfd, const void buf, size_t n, int flags,
                         const struct sockaddr *addr, socklen_t addr_len);
    ```
    ```

Crypto operations cost (no HW offloading)

	Vector of packets		Single packet	
Consumer: Signature verification				
RSA-1024	52.2us	[51.5 52.9]	140us	[132 149]
ECDSA-192	412us	[406 417]	757us	[697 817]
Producer: Signature computation				
RSA-1024	440us	[437 443]	775us	[733 818]
ECDSA-192	380us	[377 383]	701us	[661 740]
SHA-256 hash computation on MTU packet				
1.5kB	9.44us	[9.38 9.50]	28.62us	[31.03 32.08]
9kB	31.55us	[31.03 32.08]	68.26us	[63.63 72.89]

cisco