TCP/ICN: Carrying TCP over Content Centric and Named Data Networks

Ilya Moiseenko
Cisco Systems

Dave Oran
Cisco Systems
Outline

I. Introduction

II. Design
   – Basic fetching proxy
   – Reliable prefetching proxy
   – Unreliable prefetching proxy

III. Evaluation

IV. Conclusions
I. Introduction
Why Bother?

You could just run Dual-Stack in "Ships-in-the-night" mode.

But... a major attraction of ICN is simplification and performance in environments not friendly to IP:

• **Mobile edge** – eliminate tunnels, respond more quickly to mobility events, temporal caching at the edge for better error control

• **IoT** – Much smaller code footprint, better integration with radio protocols, lower background chatter than IP
High-level goals

1. Unaltered TCP/IP stack and applications
2. Preserve TCP end-to-end semantics
   – Split-TCP functionality can be added afterwards
3. Pull data between proxies (i.e. don’t require data push in Interests)
4. Compatibility with both CCN and NDN
5. Minimize overhead
Non-goals

1. Support of UDP or other transport protocols
2. Heterogeneous addressing and routing
   – Each proxy owns a routable name prefix
   – Do not propose IP addr <-> name prefix mapping
3. Path MTU-discovery and fragmentation
   – Assume standard 1500-byte TCP MSS and up to 9000 bytes ICN Data messages
II. Design
Main challenges

• How to reconcile the TCP/IP push model with the ICN pull model?
• How to minimize inflation of message count and message sizes?
• How much does the translation function need to understand the TCP state machine?
• How to marry the ICN and TCP congestion and flow control models?
Design alternatives for TPC/ICN Proxies

• We designed and evaluated three approaches:
  – Simple one-to-one mapping of TCP messages to ICN exchanges
  – Track the TCP state machine in the proxies to provide reliability over the ICN segment of the path
  – Map TCP Data to ICN Data using an unreliable inter-proxy channel

• Connection state machine same for all three – differ only in the data phase
Shared parts

TCP connection setup

TCP connection teardown
Basic fetching

- Data encapsulates a TCP segment carrying payload
  
  /[forward-proxy-prefix]/[TCP-4-tuple]/[TCP-sequence-number]/[Wraparound-number]

- Interest encapsulates an empty TCP segment
  
  /[reverse-proxy-prefix]/[TCP-IP-headers]/[nonce]

- Slow: 2x RTT, 2x packets
Reliable prefetching

Data sequencing is independent from TCP sequencing

– Impossible to predict TCP sequence number progression

TCP/IP:

SEQ # 1223682
ACK # 376523

TCP/IP:

SEQ # 1225142
ACK # 376523

TCP/IP:

SEQ # 376523
ACK # 1225142

Forward proxy

Unidirectional TCP flow

TCP 4 tuple 129
Interest: /<routable prefix>/<connection id>/<sequence#>

TCP 4 tuple 130
Interest: /<routable prefix>/<connection id>/<sequence#>

TCP 4 tuple 129
Data: /<routable prefix>/<connection id>/<sequence#>

TCP 4 tuple 130
Data: /<routable prefix>/<connection id>/<sequence#>

SEQ # 376523
ACK # 1225142
Interest: /<routable prefix>/<TCP/IP headers>/<nonce>
Reliable prefetching

- Reverse proxy transmits an Interest prior to TCP segment arrival at the forward proxy
  - Enforces reliability
- Forward proxy advertises approx. TCP cwnd size
- Faster, but unstable. A lot of state in the proxy. Other issues.
Unreliable prefetching

Time-delayed naming

– Data names match TCP sequence numbers delivered one RTT ago
Unreliable prefetching

- Interests encapsulating ACKs prefetch new TCP segments
  - Not enough ACKs due to delayed ACK TCP mechanism
  - Package multiple TCP segments in a single Data message
- No additional reliability in ICN protocol
Unreliable prefetching

A problem of **full-duplex** TCP connection:

TCP piggybacks ACKs in the data segments

--> no Interests are sent by the proxies

• A proxy detects full-duplex by analyzing ACK #

• Generates Interests on its own
  – Fixed probability (e.g. 50% = every 2\textsuperscript{nd} segment)
  – Function of ACK # progression
III. Evaluation
Evaluation

TCP cubic over ICN vs. TCP cubic over IP

- NDNsim
- Proof of concept *nix implementation
  - Reliable prefetching
Flow completion time

Unreliable prefetching is 10% slower than TCP/IP
TCP retransmissions

Total number of TCP retransmissions (packets)

Time (ms)

- Basic fetching
- Reliable prefetching
- TCP cubic
- Unreliable prefetching
TCP congestion window size

The diagram illustrates the TCP congestion window size over time for different fetching methods: Basic fetching, Reliable prefetching, and TCP cubic. The x-axis represents time in milliseconds (ms), and the y-axis represents Cwnd (bytes). The graph shows the fluctuations in the congestion window size, highlighting the differences between the methods.
TCP retransmission timeout

![Graph showing TCP retransmission timeout](image-url)

- Basic fetching
- Reliable prefetching
- TCP cubic
- Unreliable prefetching
Flow fairness and completion time

- Unreliable prefetching
- tcpperf tool + proxy PoC
- 10 competing unidirectional flows
- Within 5% difference with TCP/IP
- Fair
Beyond synthetic tests

Web browser | ICN forwarder | ICN forwarder | ICN forwarder | Web server
TCP/ICN proxy | TCP firewall | TCP/ICN proxy | TCP firewall

TLS requires additional accommodations

– Inject extra Interests at TCP connection setup phase to fetch TLS server hello messages
Conclusions

• We can run many Internet applications without IP

• TCP is highly optimized for IP
  – But it works surprisingly well over ICN!