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

Ilya Moiseenko
Cisco Systems
ilmoisee@cisco.com

Dave Oran
Cisco Systems
oran@cisco.com

ABSTRACT

Today's Internet applications and protocols are not compatible with Information Centric Networking (ICN) protocols and there is no straightforward way of rapidly switching protocol architectures. Network operators incrementally deploying an ICN infrastructure will have to provide compatibility with existing TCP/IP applications and manage co-existence of IP and ICN networks. One approach to co-existence is to allow TCP and the applications using it to work transparently over an ICN substrate instead of over IP. This paper presents a TCP/ICN proxy capable of carrying TCP traffic between TCP/IP endpoints over ICN network. The main challenge for this approach to co-existence is transforming the TCP push model to the ICN pull model. We evaluated several alternative TCP/ICN proxy designs in a simulation environment. We chose the most promising of these designs and developed a proof-of-concept *nix implementation. Performance measurements of both simulation and real implementation demonstrate that with our proxy design TCP can traverse ICN networks without significant additional delay or loss of goodput.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Architecture and Design; Network Protocols; Distributed Systems

Keywords

ICN; NDN; CCN; TCP; interoperability; co-existence

1. INTRODUCTION

Content Centric Networking (CCN) and Named Data Networking (NDN) are two related general-purpose, information-centric network (ICN) architectures [1, 2]. Both use a pull-based stateful forwarding model: clients send requests into the network to fetch data; network nodes keep the state of the forwarded requests; data replies follow the symmetric reverse path using the network state; no other unsolicited data transmission is allowed.

The change of addressing from host-based to the flexible content-based scheme and the change of the network model from push to pull are the two primary obstacles in adapting IP applications and protocols for NDN and CCN networks. At first glance it might seem that modern web applications can be easily adapted to work with ICN networks, because NDN/CCN pure pull-based communication model seems to match the request-reply mechanics of HTTP interactions. However, CCN/NDN networks do not have the protocols equivalent to TCP, TLS, etc. [3], and even if there were any, such adaptation would require changes in the application source code, library linkages, etc.

Even if ICN technology succeeds as an attractive replacement for IP as the spanning layer of the Internet, IP and ICN networks may need to co-exist for a long transition period. Co-existence of IP and ICN networks includes the following scenarios: (1) ICN endpoints communicating over an IP network, (2) IP endpoints communicating over an ICN network, and (3) an IP endpoint communicating with the peer ICN endpoint over an ICN or IP network. The ICN research community understands reasonably well how to build an ICN overlay (scenario 1) over an IP network, but the problem of legacy IP application support remains open. Since there is no straightforward way for application developers to quickly adapt existing software to ICN networks, this paper focuses on scenario 2.

A vast number of applications and protocols use TCP: web browsing (HTTP 1.1), email (SMTP/POP, IMAP), routing (BGP), DNS (optionally), etc. Therefore, it is essential for network operators deploying ICN in their infrastructure to provide backward compatibility with TCP/IP applications.

This paper examines three alternative designs for a TCP/ICN proxy capable of transparently carrying TCP traffic between TCP/IP endpoints over an ICN network. The main challenge any TCP/ICN translation proxy faces is how to reconcile the TCP/IP push model with the ICN pull model. Within this context, there are a number of associated design tradeoffs, such as how to minimize inflation of message count and message sizes, how much the translation function needs to understand the TCP state machine, whether and how to preserve TCP's true end-to-end reliable delivery guarantees, and how to marry the ICN and TCP congestion and flow control models.

The remainder of the paper is structured as follows. Section 2 discusses common protocol migration methods using IPv4-to-IPv6 transition strategies as analogous use cases. It also compares the needs of translation with the similar issues faced by TCP performance enhancing proxies. In Section 3, we describe and analyze three alternative designs for inter-proxy ICN protocols carrying TCP semantics. Section 4 provides the measurements from our simulation and proof-of-concept implementation models. In Section 5, we conclude and discuss future work.

ICN'16, September 26-28, 2016, Kyoto, Japan

DOI: http://dx.doi.org/10.1145/2984356.2984357

2. RELATED WORK

Techniques for migrating from one protocol family to another have been a source of many design paradigms over the years, however two have dominated: protocol translation and tunneling. As examples of we briefly discuss approaches in the case of migration from IPv4 to IPv6, which are very similar as opposed to the more substantial differences between IP and ICN.

In the case of IPv4/IPv6 migration, both translation and tunneling approaches have been employed. Translation is accomplished by having a special translation service rewriting IP headers on the border of IPv4 and IPv6 networks. It is possible to have stateless translation (e.g. SIIT [4]) if IPv4 addresses are deterministically mapped to IPv6 addresses. Stateful translation (e.g. NAT-64 [5]) is necessary if IPv4 and IPv6 address binding has to be resolved. Translation in the IPv4-to-v6 case can deal with all the scenarios we outlined in the introduction for co-existence because the protocol semantics are essentially unchanged.

Tunneling has also been extensively employed for IPv4/IPv6 coexistence and achieves heterogeneous traversal, for example, IPv4 traffic over an IPv6 network [6]. The basic machinery of tunneling is to have two tunnel endpoints on the borders of heterogeneous networks. When the ingress endpoint receives an IPv4 packet, it encapsulates it with IPv6 protocol header and puts the whole IPv4 packet in the payload of the new IPv6 packet. Then the IPv6 packet is forwarded through the IPv6 network. When the egress endpoint receives the IPv6 packet, it decapsulates the packet and forwards the original IPv4 packet towards the IPv4 endpoint.

While IPv4-to-IPv6 transition technologies only needed solve the heterogeneous addressing problem, IP-to-ICN transition technologies have to deal with substantially different protocol semantics. Simple tunneling cannot overcome the substantial state machine differences between TCP's sender-based push semantics and ICN's receiver-driven pull semantics. Some form of translation is needed. Conversely, translation alone often fails to maintain the level of transparency needed for the parts of the protocol unaffected by the state machine differences, hence tunneling is used for those protocol elements to ensure transparency where needed.

Other related work includes performance-enhancement proxies (PEPs) such as split TCP [7, 8]. A split TCP system terminates the TCP connection received from one TCP endpoint and establishes a corresponding TCP connection to another TCP endpoint. This is typically done to allow the use of a third connection between two PEPs, which could either be a TCP connection optimized for the link or a proprietary protocol running on top of UDP.

There is not much prior work in the area of IP-to-ICN transition and co-existence. Trossen et al. proposed to use PURSUIT ICN as an underlay for IP/TCP/HTTP-based services to achieve better utilization in HTTP unicast scenarios, better network management, and fairer content distribution [9]. We tend to agree that an ICN underlay might make some things better for IP-based services, and offer our TCP proxy approach as one promising direction to pursue.

3. TCP/ICN DESIGN ALTERNATIVES

Our design approach is to carry TCP over an ICN infrastructure by inserting a translation+tunneling proxy at the entry and exit points of the ICN network. The entry and exit proxies perform complementary functions. A *forward proxy* performs translation functions needed to express TCP semantics to the ICN network and encapsulates TCP/IP segments into Interest and Data packets for tunneling over the ICN infrastructure. A *reverse proxy* processes the ICN pull-based state machine and decapsulates Interest and Data packets back to TCP/IP segments. This section states our design goals, provides a detailed description of inter-proxy protocols, and discusses the trade-offs of each design.

3.1 High-level design goals

The objective of this work is to find the solution satisfying the following principles and goals.

3.1.1 Unaltered TCP/IP stack and applications

Any existing TCP/IP application ranging from an interactive login to high throughput secure web browsing must continue to work without modification to its source code. No modifications to either the code or configuration of the TCP/IP stack hosting the applications can be required. Additionally, the TCP/ICN proxy must work correctly with the various TCP congestion control schemes.

3.1.2 Preserve TCP end-to-end semantics

Unlike a proxy in a split TCP system that terminates a TCP connection, our goal is to carry TCP semantics correctly end to end. Split TCPs try to improve TCP performance by (among other things) pre-acknowledging TCP segments before the actual TCP reader. This can in some cases increase the throughput by reducing the feedback RTT, but takes away TCP end-to-end semantics. All TCP/ICN proxy designs discussed in this paper preserve end-to-end semantics and do not use pre-acknowledgment.[1] Therefore, we believe we can make fair performance comparisons between TCP over an ICN network and native TCP/IP.

3.1.3 Pull data between proxies

Pushing TCP data in Interest packets might seem to be the most straightforward approach, but it is at odds with the architectural tenets of CCN and NDN, and leads to several major problems. If a TCP flow is mapped to Interest-Data exchanges where TCP data is encapsulated in Interest packets and TCP acknowledgements in Data packets, it would require ICN forwarders to have specialized congestion control, fragmentation, and ARQ schemes to effectively deal with heavyweight Interest packets and lightweight Data packets. Alternatively, if a TCP flow is mapped to Interest-Interest exchanges with no closure via Data packets, then it would have the same problems with congestion control, fragmentation, and ARQ, while introducing an additional difficulty for Denial-of-Service mitigation techniques based on assessing Interest satisfaction rates [10, 11].

Therefore, for all of our design variants we follow the intended ICN principle that TCP data is requested by Interest packets and returned in Data packets. This allows our designs to benefit from useful properties of ICN networks such as one-to-one flow balance, multi-path Interest forwarding, etc., that can potentially improve the performance and robustness of TCP.

3.1.4 No requirement for longest prefix match in the PIT

The designed inter-proxy ICN protocol must use a deterministic naming scheme and cannot rely on Interests carrying incomplete names (e.g. missing TCP sequence numbers, etc.), because we desire a protocol that is compatible with both the NDN and CCN architectures.

[1] If relaxing TCP semantics is acceptable in some settings, pre-acknowledgment functionality could certainly be added.

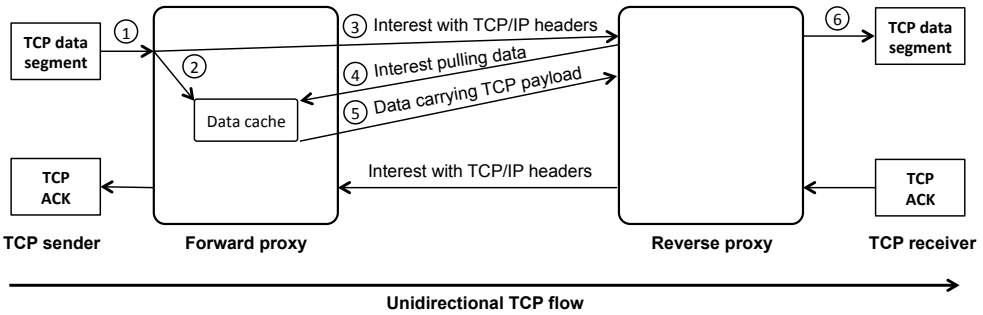


Figure 1: Basic fetching proxy design and operation.

3.1.5 Minimize overhead

We attempt to ameliorate any negative impact of the inter-proxy ICN protocol on TCP goodput by minimizing both message size and message count. To allow proxies to be low overhead and scale to large numbers of TCP connections, we minimize the amount of state and computational work the TCP/ICN proxy has to perform for each TCP connection.

3.2 Non-goals

The present work focuses on the performance aspects of TCP-over-ICN, however, it is important to mention some of the remaining co-existence and deployment related issues.

3.2.1 Support of IP or other transport protocols

The design of inter-proxy protocols relies on TCP-specific protocol headers and the specific connection-oriented features of TCP. It is potentially possible to apply similar techniques to support other connection-oriented transport protocols (e.g. SCTP), but it is not yet clear how to use such techniques for raw IP or UDP datagram protocols.

3.2.2 Heterogeneous addressing and routing

TCP applications continue using their usual destination and source IP addresses and ports. Network operators can redirect TCP/IP traffic towards TCP/ICN proxies using such standard techniques as NAT, SOCKS [12], etc.

TCP/ICN proxies are similar to PEPs in maintaining some amount of state about the ongoing TCP connection and in the necessity of locking the TCP connection to a specific proxy. In order to consistently forward Interest and Data messages between the appropriate proxies, TCP/ICN proxies must own routable ICN names which are used as name prefixes for outgoing Interest and Data messages. It is possible to use name prefixes that are algorithmically mapped to the IP addresses. Another option is to have a name prefix lookup process similar to the reverse DNS lookup in order to match a destination IP address to the prefix of the corresponding proxy.

3.2.3 Path MTU-discovery and fragmentation

We assume that TCP/ICN proxies enforce standard 1500-byte MSS for TCP/IP applications and can use jumbo-sized ICN Data packets (e.g. 8KB). Our designs may in some cases not work well for cases needing path MTU discovery over heterogeneous networks.

3.3 Basic fetching proxy

A "Notify-then-pull" mechanism realized via Interest-Interest-Data packet exchange is the basic way of emulating push semantics over an NDN / CCN network. An entity willing to push the data notifies another entity that data is ready to be pulled. Part of the design therefore is adherence to specific naming conventions so that names are predictable for both proxies processing the packets.

The primary benefit of a TCP proxy design based solely on a "notify-then-pull" mechanism is its simplicity. Both proxies maintain only the minimum amount of state about each TCP connection, and are not required to run any complicated time- or state-synchronized inter-proxy protocol or handle any of the special cases that arise in the TCP state machines.

3.3.1 Inter-proxy protocol operation

The transcript of protocol operation for the basic fetching proxy corresponds to the numbers in the Figure 1.

1. When a TCP segment arrives at the forward proxy, the proxy checks whether the segment is carrying a payload. In such case, the information in TCP/IP headers is used to construct the name of a new Interest packet.[2] The original TCP/IP segment is placed in the payload of a corresponding new Data packet (i.e. encapsulated in ICN Data packet).

2. We use the following naming convention for Data packets carrying TCP segments with payload:

   `/[forward-proxy-prefix]/[TCP-4-tuple]/[TCP-sequence-number]/[Wraparound-number]`

   `forward-proxy-prefix` is the routable name prefix of the proxy that has received TCP segment. The `TCP-4-tuple` name component(s) includes source IP, source port, destination IP and destination port, which are necessary for TCP connection identification by the proxy. The `TCP-sequence-number` name component exactly matches the sequence number of the corresponding TCP segment. Since the binding of data to names in NDN and CCN is not mutable, `Wraparound-number` is necessary to prevent name collisions when the TCP sequence number wraps around. We considered the use of TCP timestamps for versioning purposes, but preferred sequential versioning scheme, because TCP timestamps can be disabled on some hosts. In a sequential versioning scheme, both proxies detect wraparound events independently and increment the wrap-around number each time it occurs.

3. We use the following naming convention for Interest packets carrying only TCP/IP headers with no payload:

   `/[rev-proxy-prefix]/[TCP-IP-headers]/[nonce]`

   `rev-proxy-prefix` is the routable name prefix of the reverse proxy decapsulating Interest and Data packets into TCP/IP segments. `TCP-IP-headers` carry all TCP/IP headers including all options. `nonce` is a random number that ensures

[2] In the CCN architecture, TCP/IP headers can be placed in the Interest payload field instead of the name.

the uniqueness of the Interest necessary for preventing ICN forwarders from suppressing duplicate ACK transmissions sent by TCP receivers. Instead of a nonce one could use the signature of the Interest for uniqueness. This would provide additional protection of the inter-proxy protocol against malicious senders generating 'fake' Interests in order to disrupt the operation of the protocol.

4. If an Interest arriving at the reverse proxy carries TCP/IP headers indicating there is accompanying TCP data to be pulled, the reverse proxy transmits a 'reflexive' Interest packet with the name that is guaranteed to match the Data packet stored in the cache of the forward proxy. The reverse proxy does not enforce reliability, therefore TCP endpoints continue to be responsible for recovering data loss, just as they do when running over IP.

5. When a forward proxy receives the reflexive Interest packet, it attempts to find matching Data packet in its Data cache.

6. When the reverse proxy receives a Data packet, it transmits the decapsulated TCP segment over IP to the TCP receiver.

3.3.2 Quick analysis of the protocol

This basic protocol does not demand a lot of memory or processing resources, but it is unreasonable to expect good TCP performance, because the number of packets and round-trips are both doubled. Potential benefits of Data packet caching at intermediate forwarders had not fully materialized as discussed in Section 4.

Connection setup. The TCP three-way handshake maps to an equivalent three-way Interest exchange (Figure 2) with both proxies going through a series of initialization states.[3] It is beneficial to use only Interests in the TCP setup phase, because the corresponding Data messages can be saved for the purpose of delivering the actual TCP data.

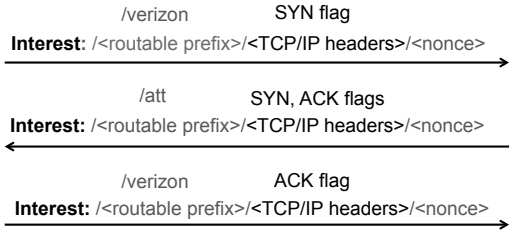


Figure 2: ICN message sequence for TCP connection setup.

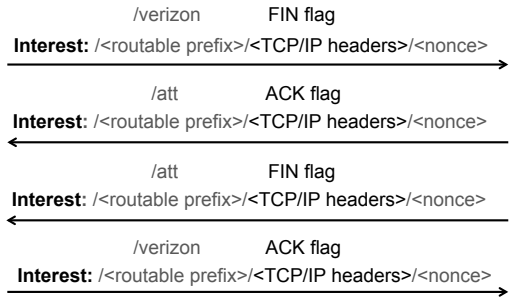


Figure 3: ICN message sequence for TCP connection teardown.

[3]TCP "Christmas tree" segments or single segment sessions (i.e. SYN/Data/FIN/ACK) are not supported by our protocol at this time.

Connection teardown. Analogously, the TCP 4-way teardown maps to an equivalent 4-way Interest exchange (Figure 3) with both proxies going through a series of termination states.

It is important to note that the same Interest exchanges for TCP connection setup/teardown are used in other inter-proxy ICN protocols described in this paper.

Full duplex connection. In the unidirectional data transfer scenario, each TCP Data-ACK exchange results in an Interest-Interest-Data-Interest exchange. In the bidirectional data transfer scenario, each TCP Data-Data exchange results in an Interest-Interest-Data-Interest-Interest-Data exchange leading to 3x packet count expansion over native TCP/IP.

3.4 Reliable prefetching proxy

The results of experimentation with the basic proxy design motivated us to look for a more advanced solution with a more efficient inter-proxy control protocol. One improvement is for the reverse proxy to maintain a window of outstanding Interest packets in such a manner that whenever the forward proxy receives a TCP data segment there is an outstanding Interest at the proxy which could be matched with ICN Data packet for the TCP segment in response. Compared to basic fetching, prefetching reduces latency and decreases the number of packet transmissions. This scheme is 'reliable' in the sense that the reverse proxy ensures successful delivery of all requested Data packets from the forward proxy. Reliable prefetching can also conceal packet losses in the ICN network infrastructure from TCP endpoints, which can potentially improve TCP throughput by avoiding TCP retransmissions.

In the basic fetching protocol, data names are mapped to TCP sequence numbers. In the reliable prefetching protocol, data names cannot be mapped straightforwardly to TCP sequence numbers because the reverse proxy cannot predict the progression of TCP sequence numbers of available Data segments at the moment when it has to send Interest packets.[4] Instead, Data names contain an independent (of the TCP sequence space) sequence name component whose value monotonically increases for each new incoming TCP segment (Figure 4). Such a predictable naming pattern allows reverse and forward proxies to have a synchronized understanding of what packets had been produced, fetched, lost and retransmitted.[5]

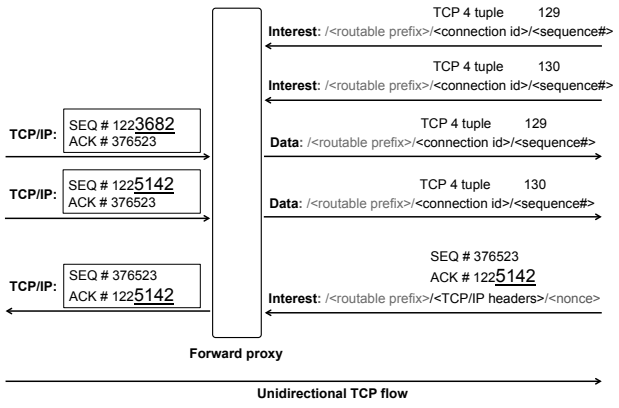


Figure 4: A snapshot of ICN message sequence numbering scheme used in the reliable prefetching protocol.

[4]It might be possible to predict TCP sequence numbers for bulk transfers, because TCP segments are MSS-sized, however such techniques will fail in other circumstances.

[5]As stated in our design goals, incomplete Interest names cannot be used, because we do not wish to depend on PIT LPM.

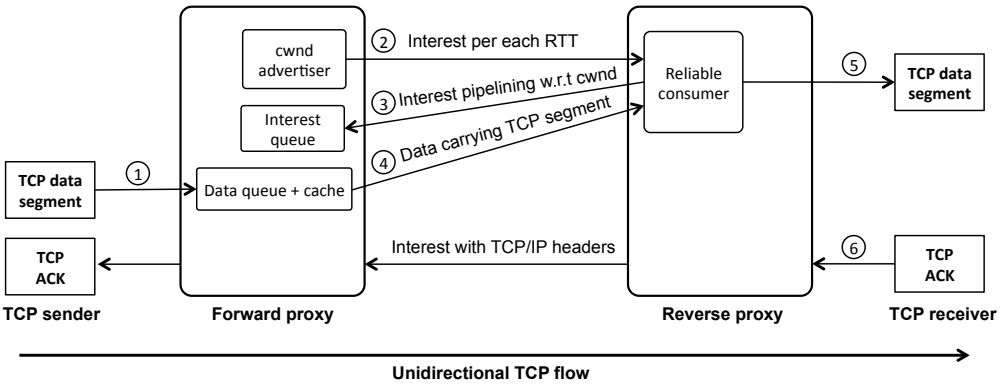


Figure 5: Reliable prefetching proxy design and operation.

3.4.1 *Inter-proxy protocol operation*

The transcript of protocol operation for the reliable prefetching scheme corresponds to the numbers in the Figure 5.

1. If an incoming TCP segment contains a payload, the whole TCP segment is encapsulated in a Data packet carrying a name with monotonically increasing sequence number. The Data packet is placed in the data queue if there is no matching Interest in the Interest queue.

   `/[forward-proxy-prefix]/[TCP-4-tuple]/[sequence-number]`

2. Efficient operation of the reliable prefetching protocol largely depends on accurate estimation of the consumer's Interest window size. Too many outstanding Interests, possibly retransmitted multiple times because of being sent too early, consume bandwidth and processing capacity. Too few outstanding Interests will depress the TCP transmission rate by capping the outstanding Interest count below the TCP congestion window size. To give a hint to the consumer at the reverse proxy, the forward proxy can periodically advertise the estimated congestion window size of the TCP sender. A number of passive and active TCP parameter estimation heuristics can be found in the literature [13, 14]; we used a rough estimation of congestion window size available in the proxy by observing TCP segment arrivals, as described below.

   In the absence of a way to dynamically adjust the outstanding Interest count, as the TCP sender's congestion window size increases so will the TCP segment data queue at the forward proxy. The size of TCP data queue can therefore act as surrogate to estimate the TCP congestion window size. By advertising this periodically (e.g. once per RTT) the reverse proxy can track the congestion window with a one-RTT delay and adjust the consumer's outstanding Interest window.

3. The consumer at the reverse proxy performs transmission of new Interest packets and retransmission of timed out Interests. This scheme allows the ICN network to conceal packet losses from TCP endpoints by using built-in transient caching of Data packets in the NDN / CCN infratructure.

4. Incoming Interest packets are placed in the Interest queue if there is no matching Data packet in the TCP data queue. Otherwise, the Data packet is returned immediately to the reverse proxy and may be cached at intermediate forwarders, allowing faster recovery from loss.

5. The TCP data segment is extracted from the Data packet and transmitted to the TCP receiver over IP.

6. TCP segments that do not carry any payload (e.g. ACK, SYN, FIN packets, etc.) are transformed into Interest packets and do not have to be fetched separately. The namespace design for Interest packets is identical to the one in the basic fetching proxy.

3.4.2 *Quick analysis of the protocol*

The reliable prefetching protocol has an identical connection setup, teardown, and ACK delivery mechanism to the basic fetching protocol. Only the data delivery method differs. Instead of an Interest-Interest-Data-Interest exchange for each TCP Data - ACK exchange, the reliable prefetching protocol uses normal Interest-Data exchange, using an independent packet sequence numbering scheme and with periodic advertisements of the estimated congestion window size at the TCP sender.

Idle TCP sender. When the TCP sender is idle for more than a few RTTs, the TCP data queue at the forward proxy becomes empty. Through periodic advertisements, the reverse proxy infers that the number of outstanding Interests should be reduced to zero and only resumed when the TCP sender resumes transmission.

Full duplex connection. TCP Data-Data exchanges are realized through both TCP/ICN proxies performing complementary steps 1 — 5 of the protocol description above with exception of step 6 due to the absence of standalone TCP ACK packets.

Selective acknowledgements. The reliable prefetching ICN protocol cannot take advantage of TCP SACK, because ICN messages are numbered in a separate sequence space from the TCP sequence space, and there is no reliable way of transforming one sequence number to another one as discussed earlier.

On paper, reliable prefetching decreases the number of transmitted packets from 4 to (3 + 1/RTT) per each TCP Data-Ack, decreases the latency of communication, and enables fast recovery due to the transient caching at intermediate forwarders. With our simulation model we found out that while reliable prefetching has better performance than basic fetching, having a separate TCP-like control loop inside the original TCP control loop leads to higher RTT, RTO and congestion window variation at TCP senders. Section 4 discusses these effects in more detail. It is also likely that the inner control loop negatively affects TCP flow fairness.

The core problem with the reliable prefetching protocol is inexact matching of the TCP congestion window size to the Interest window size at the reverse proxy's consumer, which in turn causes a large number of Interest retransmissions. Long lived Interests in the range of several seconds could reduce the number of unneces-

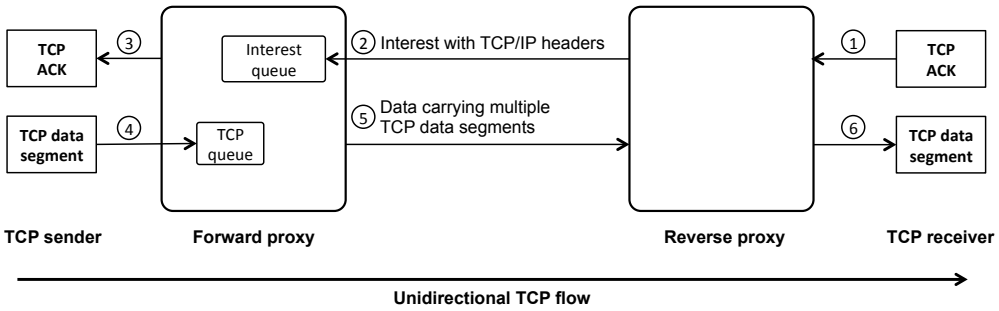


Figure 6: Unreliable prefetching proxy design and operation.

sary retransmissions, but cannot be used in practice, because each potential loss of a long lived Interest disrupts the TCP sender and decreases its congestion window size. As a rule of thumb, an Interest retransmission timeout ought to be smaller than TCP RTO at all times.

Another issue with the reliable prefetching protocol occurs when the TCP sender detects loss and retransmits a TCP segment. When TCP endpoint retransmits a packet, the forward proxy creates new Data packet with new sequence number. This leads to the situation where NDN / CCN network segment has more circulating packets than the IP network segment, causing significant overhead due to the fact that all of these packets are reliably delivered. A potential solution for this problem is to make the forward proxy remember all processed TCP sequence numbers and drop retransmitted packets. However, this causes data transfer halts in the situation when (1) ACK packet (Interest) is lost in the NDN / CCN infrastructure, (2) the TCP sender attempts retransmission. The forward proxy would drop such a packet since it has already processed and cached the packet with that TCP sequence number. Conversely, the reverse proxy would be certain that it has already fetched a Data packet with the matching NDN/CCN sequence number. Handling such situations would likely require acknowledging ACKs or notifying the reverse proxy about TCP retransmissions, which furthermore increases complexity of the system.

3.5 Unreliable prefetching proxy

Simulation of the reliable prefetching protocol demonstrated both its advantages over non-prefetching technique and the issues inherent in the interplay of TCP retransmission behavior with inter-proxy protocol machinery. A possible way of eliminating the complexity of Interest window management is to clock the prefetching process exclusively by the receipt of acknowledgements from the TCP receiver. In other words, is it possible to transmit an Interest only in the case when there is an incoming ACK packet from the TCP endpoint? Is it possible to eliminate all independent Interest retransmissions and let TCP or in-network Interest retransmission to react to the losses instead of adding reliability to the inter-proxy protocol?

Several problems present themselves as soon as we start working through the protocol operations. First, TCP usually acknowledges only every other TCP segment due to the ubiquity of delayed ACK optimizations. Second, the TCP sender is constantly probing for more available bandwidth by increasing its congestion window size. In normal circumstances the number of TCP acknowledgements is usually smaller than the number of TCP data segments. Therefore, we investigated two ways to overcome the deficit of Interests in order to accommodate all incoming TCP data:

- artificially inflate the number of transmitted Interests. For example, each empty TCP ACK could be duplicated by the reverse proxy as multiple Interests, and the forward proxy would de-duplicate Interests to avoid invalid duplicate ACK transmission towards the sender.
- package multiple TCP data segments in a single Data packet to match the number of ACKs. This requires the NDN / CCN protocol MTU to be a multiple of the negotiated TCP MSS.

As an engineering tradeoff, we concluded that packaging multiple TCP data segments is the "lesser evil"; a proxy enforcing standard 1500 byte MSS to the TCP endpoints and operating with 8KB Data packets potentially encapsulated in IP jumbo frames are not too unrealistic assumptions to make. The other approach of inflating the number of Interest transmissions adds more complexity to the system and has non-deterministic behavior.

3.5.1 Inter-proxy protocol operation

The transcript of protocol operation for the unreliable pre-fetching proxy protocol corresponds to the numbers in the Figure 6.

1. A TCP segment not carrying any payload (e.g. empty ACK, SYN-ACK, etc.) is encapsulated into an Interest according to the same naming convention used in the basic fetching model.
2. Incoming Interests are stored in the Interest queue until their expiration.
3. If an Interest indicates that there is no accompanying TCP data (e.g. empty ACK) then the proxy immediately decapsulates the TCP segment from the Interest and transmits it to the TCP endpoint.
4. The TCP sender receives an ACK and transmits new data segment(s), which arrive at the forward proxy. The forward proxy puts the TCP data segment(s) in the TCP queue and determines how many TCP segments must be taken from the queue to be packaged together in a single Data packet.
5. A Data packet is created with a name that matches the name of the oldest non-expired Interest from the Interest queue (i.e. head-of-line).
6. The proxy receives a Data packet and passes the multiple TCP data segments it contains to the TCP receiver over IP.

3.5.2 Quick analysis of the protocol

Unreliable prefetching is simpler and more scalable than reliable prefetching, because it does not have an independent control loop. The protocol does not involve transmission of any extra packets, and in fact the number of Interest and Data packets is lower than the number of packets in the IP network segment. This helps to amortize ICN related overhead such as long names, packet signature, etc.

Idle TCP sender. Unreliable prefetching is subject to the problem of an idle TCP sender. When a TCP sender becomes idle for a relatively long period of time all Interests in the Interest queue in the forward proxy expire, so that when TCP sender transmits new data the forward proxy does not have outstanding Interests in its queue. This issue is resolved by having a "wake up" mechanism based on an Interest-Interest-Data exchange that restarts the normal operation of the protocol.

No application level caching. The naming model uses the original TCP sequence numbers similarly to the basic fetching protocol. To allow intended duplicate ACKs to reach TCP endpoints and avoid aggregation in NDN / CCN forwarders, and to prevent any possible re-use of previously used names with modified content (i.e. mutable data) every Interest name has a random value (nonce) appended to make it unique. The negative side effect of that is the inability to leverage application level caching in the NDN / CCN network, however, fast Interest retransmission by the forwarder is still viable as a recovery technique.

Time-delayed naming. An interesting observation is that the names of Data packets do not reflect their content in a meaningful way. At best, the relationship between the content (i.e. TCP data segments) and the names can be characterized as time-shifted, because names contain the sequence numbers used by TCP one round-trip before. Figure 7 is a snapshot of the ICN message sequence numbering scheme at the moment when a forward proxy receives an acknowledgement Interest for previous TCP segments and uses it to transmit a Data message carrying two new TCP data segments.

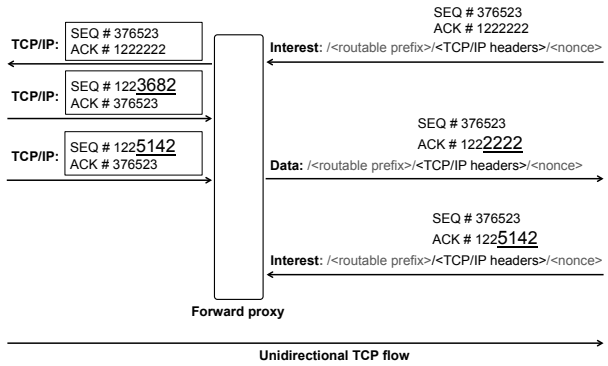
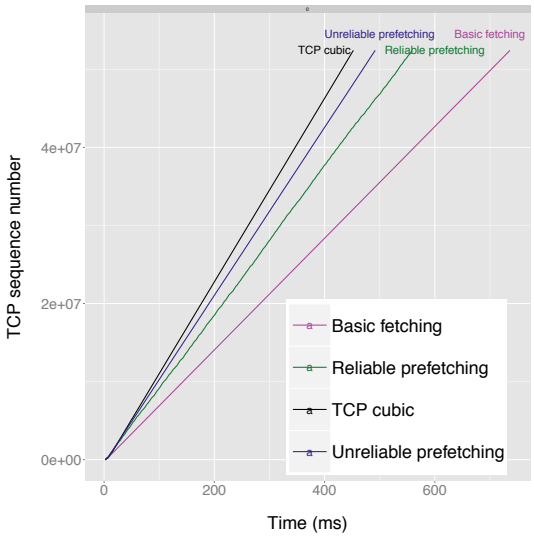


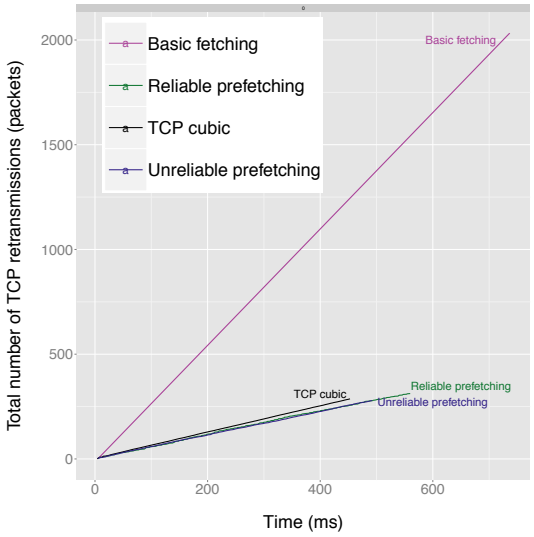
Figure 7: A snapshot of ICN message sequence numbering scheme using 1500-byte TCP MSS.

Full duplex connection. TCP piggybacks acknowledgements in its data segments, which would result in no Interests sent by TCP/ICN proxies without a special handling of full duplex transfers. In order to recognize a bidirectional data transfer, each TCP/ICN proxy keeps track of the highest acknowledgement number received from the TCP endpoint up to the moment. If an incoming TCP data segment carries a payload and has an ACK number higher than previously recorded one, the TCP/ICN proxy must make a decision whether an Interest should be sent towards the reverse proxy to fetch the TCP data segments going in the opposite direction. The decision could be as simple as "send Interest for each TCP data segment", but that is likely to result in too many Interests reducing available bandwidth. Other approaches include:

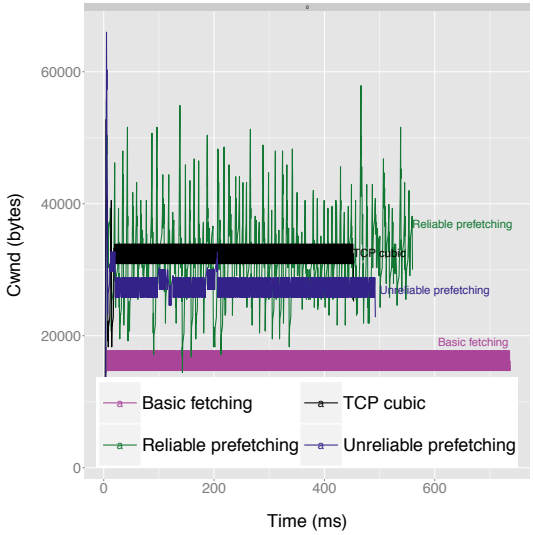
- Fixed probability (e.g. send an Interest message for every other TCP data segment – 50% probability)
- Function of the increase of acknowledgement number (e.g. send an Interest message after the acknowledgement number advances by a specified number of bytes.)



(a) TCP flow completion time.

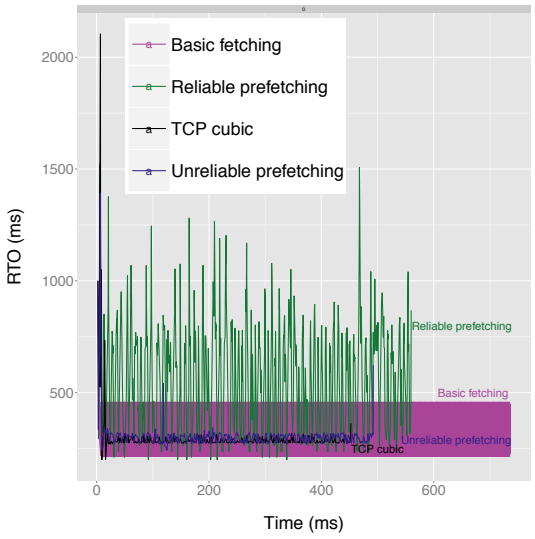


(b) TCP sender retransmissions.



(c) TCP congestion window size.

Figure 8: Unidirectional TCP transfer of 50 MB over 1 Mbps 0% loss links.



(d) TCP retransmission timeout.

Figure 8: Unidirectional TCP transfer of 50 MB over 1 Mbps 0% loss links.

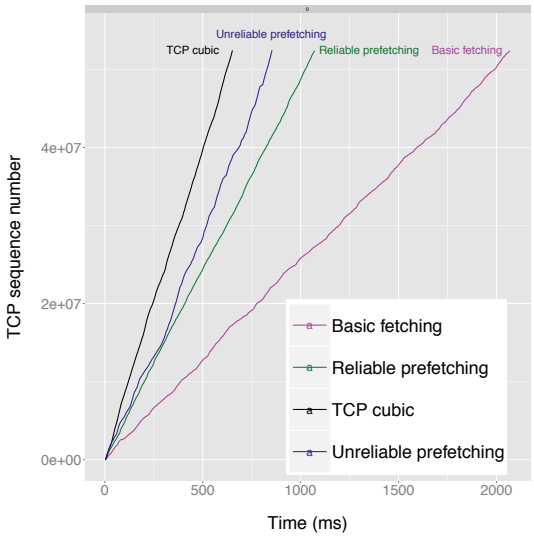


Figure 9: Flow completion time of 50 MB unidirectional TCP transfer over 1 Mbps 1% loss links.

4. EVALUATION

We built a simulation model for each TCP/ICN inter-proxy protocol to understand the performance and system complexity characteristics, and test various edge conditions (e.g. idle sender). We chose the best performing inter-proxy protocol to build as a proof-of-concept Linux/Unix implementation to explore the practical engineering trade-offs.

4.1 Simulation

We used NDNsim 2.1 [15] for rapid prototyping and ease of measuring of the inter-proxy protocols. NDNsim has both a mature NDN stack and a comprehensive TCP/IP implementation with various congestion control algorithms in its NS-3 foundation. In order to drill into the details of the protocol performance, our simulations compare the performance of a single NS-3 TCP cubic connection between client and server applications connected through either:

- a simple three-hop IP network (Figure 10a)
- a three-hop NDN network (Figure 10b) with a node hosting a forward proxy and NDN forwarder, a second node hosting only an NDN forwarder, and a third node hosting a reverse proxy and NDN forwarder.

We computed four different metrics over two types of simulation runs: 1) 1 Mbps links with 0% packet loss 2) 1 Mbps links with 1% packet loss. The most important metric from the user's perspective for most TCP flows is the flow completion time (FCT). We assess sender retransmissions to understand whether TCP performance is negatively impacted by higher RTTs or packet drops / timeouts. Tracking Instantaneous TCP congestion window size and TCP retransmission timeout (RTO) metrics allows us to detect any potential TCP flow instabilities caused by the inter-proxy ICN protocol.

Analyzing the results of 0% and 1% packet loss simulation runs in Figures 8 and 9, we can see that the basic fetching inter-proxy protocol slows down TCP cubic by at least 50% (Figure 8a). The primary reason for the slowdown is that TCP retransmissions (Figure 8b) are limiting the congestion window size. The root cause is the doubled number of packets in the ICN network for the basic inter-proxy approach. TCP RTO is high at all times due to the 2RTT Interest-Interest-Data exchanges in the basic inter-proxy protocol.

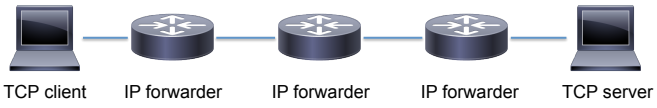
The reliable prefetching protocol has much better flow completion times due to the reduction of TCP retransmissions (Figure 8b). Figures 8c and 8d show that this protocol exhibits instability of congestion window size and retransmission timer. We failed to adequately synchronize the TCP control loop with the inner prefetching loop in spite of significant effort spent on tuning the prefetching protocol. This is a major issue, and while it might be possible to achieve higher level of synchrony between the two control loops, we have concluded that this design does not have much promise for good performance and robust behavior under varying network conditions.

The simulations show conclusively that the unreliable prefetching ICN protocol demonstrates superior performance across all four metrics. Its flow completion time is within 8% - 10% of that achieved by TCP cubic over the IP network; the number of retransmissions is almost identical to TCP/IP; the congestion window size is high and stable at all times; TCP RTO measurements are stable and almost identical to TCP/IP. The only negative side-effect of the unreliable prefetching ICN protocol appears with packet losses in ICN network segment. Figure 9 shows that unreliable prefetching falls behind TCP cubic more significantly than in Figure 8a . Since the protocol packages multiple TCP data segments together, a loss of a single Data packet results in a loss of multiple TCP segments. However, such effects can be mitigated with in-network Interest retransmission by NDN/CCN forwarders.

4.2 Proof of concept software

Building an actual TCP/ICN proxy for Linux/Unix operating systems is a good way of verifying the correctness of simulation model, obtaining more realistic performance measurements, and discovering new engineering problems not evident in simulation environment. Our experimental network topology is identical to the linear topology in NDNsim: two TCP/IP endpoints connected through three Cisco ICN forwarder nodes, two of which have forward and reverse TCP proxies running (Figure 10b).

While our simulation models could handle only a single TCP



(a) TCP/IP network setup.

(b) TCP/ICN network setup.

Figure 10: IP and ICN simulation / proof-of-concept network topologies.

connection, our proof-of-concept unreliable prefetching TCP/ICN proxy for *nix OS is capable of maintaining hundreds of simultaneous TCP connections. To ensure that the TCP/ICN proxy does not affect TCP flow fairness, we conducted a number of experiments with Iperf2 tool used to generate unidirectional and bidirectional, single and multiple competing TCP data streams between the two outermost nodes in a linear 5-node topology.

Figure 11 illustrates standard deviation of flow completion time of 10 simultaneous unidirectional short (1Mb), medium (10Mb) and large (100Mb and 500Mb) TCP flows generated with Iperf2 tool. Error bars represent the distribution of flow completion time among competing TCP flows. The compact distribution of flow completion time means that each TCP flow is treated fairly by TCP/ICN proxies regardless of TCP flow size. The similarity of mean values of flow completion time across TCP/IP and TCP/ICN network setups shows that a TCP data stream can be pulled over NDN and CCN networks without significant additional delay or loss of goodput (e.g. within 5% difference).

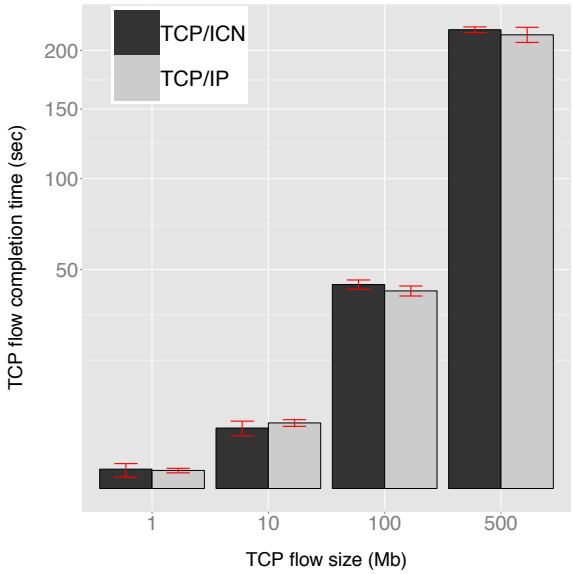


Figure 11: Flow fairness and completion time of 10 competing unidirectional TCP streams.

In addition to synthetic tests with Iperf2 tool, the TCP/ICN proxy was tested with real user traffic using the Firefox browser accessing popular web search engines, video streaming and other websites. Initially, we experienced multiple performance and functional challenges, because the major part of web interaction takes place over encrypted TLS channels that require special TLS handshakes to each participating web server, and sometimes dozens of TLS handshakes per complex web page. The root of the problem is in TLS server hello messages carrying TLS crypto information along with server certificate. TLS server hello messages occupy multiple TCP data segments for which there are not enough outstanding Interests at the proxies immediately after a successful TCP handshake. In order to facilitate TLS handshake as well as TCP implementations with fast start/open mechanism, forward and reverse proxies must transmit a certain number of additional Interests towards each other during TCP connection setup phase. This mirrors the experience in native TCP where good performance for TLS sessions is contingent on the TCP initial window being increased from 2 to 10.

5. CONCLUSION AND FUTURE WORK

Content Centric and Named Data Networking technologies have been under design and active development for almost 7 years to date. Now is the time to begin active work on realistic IP-to-ICN transition and co-existence strategies, and this paper represents one of our initial steps in this direction.

We do not know yet how to provide workable interoperability for every kind of IP application in a way that preserves the spirit and useful properties of CCN and NDN networks. For these architectures to make good on their promise, exploiting rather than avoiding their important properties seems critical. These include embracing the pull model rather than forcing push operations into the design, improving congestion and flow control through the hop-by-hop stateful forwarding model, and achieving both fairness and high network utilization through multi-path forwarding. By exploring the performance and complexity tradeoffs in achieving interoperability at the transport layer and above, we can make more intelligent decisions about where to bend or otherwise modify the ICN protocol designs to better accommodate interoperability without compromising their promise as the spanning layer of a future Internet.

The solutions proposed in this paper enable the transit of all of the original information in TCP/IP headers and packet payload between each pair of TCP/IP endpoints, thus supporting all TCP flags and options, and preserving TCP end-to-end semantics. TCP/ICN proxies do not modify the TTL value in the IP header, and therefore, from the viewpoint of the TCP endpoint, the whole ICN network segment is observed as a single potentially congested link. It is expected that the congestion in ICN network segment is controlled by TCP/ICN proxies along with actively participating NDN/CCN forwarders. This was one of the main motivations for us for avoiding excessive overhead in ICN packet count / size compared to TCP/IP packet count / size.

One very attractive target for near-term interoperability is the mobile edge, where ICN's demonstrated advantages in mobility management are most apparent. With this technology, the mobile edge could run ICN natively, with the TCP/ICN proxies in the mobile device at one end (using standard proxy daemon methods) and the other end at the Packet Core gateway in the mobile operator's network.

Beyond simply achieving interoperability, an interesting idea to explore is to understand whether there are any real benefits of using ICN as an underlay for TCP/IP traffic. It might be possible to reduce some of the overhead of the inter-proxy ICN protocol and even get ahead of the original TCP/IP in terms of goodput (especially over lossy channels) if CCN/NDN forwarders have in-network congestion management or in-network loss recovery mechanisms.

6. ACKNOWLEDGEMENTS

The authors thank the ACM ICN reviewers and our shepherd Craig Partridge for the comments that helped to improve the paper.

7. REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of CoNEXT*, 2009.

[2] L. Zhang et al., "Named Data Networking (NDN) Project," Tech. Rep. NDN-0001, October 2010.

[3] I. Moiseenko, M. Stapp, and D. Oran, "Communication patterns for web interaction in named data networking," in *Proc. of ACM ICN*, 2014.

[4] E. Nordmark, "Stateless IP/ICMP translation algorithm (SIIT)," 2000.

[5] M. Bagnulo, P. Matthews, and I. v. Beijnum, "Stateful NAT64: Network address and protocol translation from IPv6 clients to IPv4 servers," 2011.

[6] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, "Transition from IPv4 to IPv6: A state-of-the-art survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1407–1424, 2013.

[7] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*. IEEE, 1995, pp. 136–143.

[8] J. Border, J. Griner, G. Montenegro, Z. Shelby, and M. Kojo, "Performance enhancing proxies intended to mitigate link-related degradations," 2001.

[9] D. Trossen, M. J. Reed, J. Riihijarvi, M. Georgiades, N. Fotiou, and G. Xylomenos, "IP over ICN-The better IP?" in *Networks and Communications (EuCNC), 2015*. IEEE, 2015, pp. 413–417.

[10] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest flooding attack and countermeasures in Named Data Networking," in *IFIP Networking Conference, 2013*. IEEE, 2013, pp. 1–9.

[11] A. Compagno, M. Conti, P. Gasti, and G. Tsudik, "Poseidon: Mitigating interest flooding DDoS attacks in named data networking," in *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*. IEEE, 2013, pp. 630–638.

[12] M. Leech, "SOCKS protocol version 5," 1996.

[13] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP connection characteristics through passive measurements," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2004, pp. 1582–1592.

[14] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 4, pp. 1311–1324, 2014.

[15] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3," Technical Report NDN-0028, NDN, Tech. Rep., 2015.

.