Optimizing Bi-Directional Low-Latency Communication in Named Data Networking

Mishari Almishari[♭*], Paolo Gasti[‡], Naveen Nathan[#], Gene Tsudik[#†]
[♭]King Saud University, Saudi Arabia
[‡]New York Institute of Technology
[#]University of California, Irvine
mialmishari@ksu.edu.sa, pgasti@nyit.edu, nnathan@ics.uci.edu, gts@ics.uci.edu

ABSTRACT

Content-Centric Networking (CCN) is an alternative to today's Internet IP-style packet-switched host-centric networking. One key feature of CCN is its focus on content distribution, which dominates current Internet traffic and which is not well-served by IP. Named Data Networking (NDN) is an instance of CCN; it is an on-going research effort aiming to design and develop a full-blown candidate future Internet architecture. Although NDN's emphasizes content distribution, it must also support other types of traffic, such as conferencing (audio, video) as well as more historical applications, such as remote login and file transfer.

However, suitability of NDN for applications that are not obviously or primarily content-centric. We believe that such applications are not going away any time soon. In this paper, we explore NDN in the context of a class of applications that involve low-latency bi-directional (point-to-point) communication. Specifically, we propose a few architectural amendments to NDN that provide significantly better throughput and lower latency for this class of applications by reducing routing and forwarding costs. The proposed approach is validated via experiments.

1. INTRODUCTION

Today's Internet is an effective platform for a multitude of diverse applications, including: WWW, Email, P2P and VoIP. Its main architectural pillar is IP [1], which follows the host-centric packet-switched communication paradigm, where each host is referred via one or more interface addresses and communication is performed via IP datagrams. Although this model has lasted for decades, exceeding all expectations, it is starting to fray. The Internet is being used to distribute greatly increasing amounts of digital content. This unprecedented and lasting growth spurt is due to the proliferation and popularity of multimedia content, social networks as well as increasing amounts of user-generated content. The resulting fundamental change in the nature of Internet traffic has exposed limitations of the current IP-based architecture. To this end, some projects aiming to design candidate next-generation Internet architectures started within the last several years.

Named-Data Networking (NDN) [2] is one such effort that exemplifies the Content-Centric Networking (CCN) approach [3, 4, 5]. NDN explicitly names content instead of physical locations, such as hosts or network interfaces. Instead of a conversation-style semantics of IP where hosts directly address each other, NDN applications request content via a human-readable name; the network is in charge of locating and returning the closest copy of requested content. (See Section 2 for more NDN details.) NDN also stipulates that each piece of named content must be digitally signed by its producer. This allows decoupling of trust in content from trust in entities that might store and/or disseminate that content.

NDN is primarily oriented towards efficient large-scale content distribution. it is unclear how it would fare in the context of applications that do not fit the content distribution paradigm. In order to become a viable alternative to IP, NDN must also provide evidence of support for other types of Internet traffic. In other words, overall practicality of NDN depends, among other things, on how it performs outside its *forte*.

Recent and on-going research has shown that NDN offers significant advantages compared to IP in terms of performance, security and functionality (in particular, naming) with respect to real-time [6], group [7] and anonymous communication [8]. This paper focuses on one specific type of Internet traffic that exhibits characteristics very different from content distribution. It corresponds to a class of applications that involve low-latency bi-directional synchronous communication, such as audio and/or video conferencing as well as more legacy applcations such as remote login. As shown in [6] voice conferencing is feasible when NDN is used as an IP overlay. While we do not claim that this application class is not accommodated by NDN, we believe that NDN is not particularly well-suited these needs. This motivates us to explore add-on techniques that might offer better performance. As we show below, simple amendments that retain basic NDN features (and do not affect content distribution-type traffic) result in markedly improved end-to-end throughput and bandwidth utilization. This assertion is supported by experiments.

Organization. After a brief overview of NDN in the next section, we provide some motivation for optimizing bi-directional communication in NDN, in Section 3. Then, in Section 4, we describe an interest-bundle scheme that, while leaving key NDN features intact, offers markedly better performance for low-latency point-to-point bi-directional communication. We then discuss, in Section 5, NDN modifications that support our design. Section 6 reports on experimental results that confirm claimed performance gains. We overview related work in Section 7 and conclude in Section 8.

2. NDN OVERVIEW

NDN supports two types of messages: *interests* and *content packets* [9]. A content packet contains a human-readable name, actual data (content) and a digital signature computed by the content producer over the packet. Names are hierarchically structured, e.g. `/usa/cnn/frontpage/news` where "/" is the boundary between name components. An interest packet contains the name of the content requested or prefix of such a name, e.g. `/usa/cnn/` is a prefix of `/usa/cnn/frontpage/news`. In case of multiple content matching a given name prefix, optional control informa-

[*]Work done in part while at UC Irvine.

[†]The authors were supported by the NSF under project CNS-1040802 – "FIA: Collaborative Research: Named Data Networking (NDN)".

tion can be carried within the interest to restrict the desired content. Content signatures provide integrity and data origin authentiction. However, trust management (particularly, the relationship between keys and name prefixes) is the responsibility of the application.

All NDN communication is receiver-driven: a consumer initiates communication by sending an interest for a specific content. NDN routers forward this interest towards the content producer responsible for the requested name, using name prefixes (instead of today's IP prefixes) for routing. *Forwarding Information Base* (FIB) is a lookup table used to determine interfaces for forwarding incoming interests, and contains [*name_prefix*, *interface*] entries. Multiple entries with the same *name_prefix* are allowed, supporting multiple paths under which a given *name_prefix* namespace is reachable. Akin to an IP forwarding table, FIB can be populated either by a routing protocol or manually.

When a producer needs to *push* some content, it cannot unilaterally do so. Instead, it must send an interest to the intended receiver, who in returns responds with one or more interests for the content. The consumer can also acknowledge the interest from the producer with a content packet.

Each NDN router maintains a Pending Interest Table (PIT) – a lookup table containing outstanding [*interest*, *arrival-interfaces*] entries. The first component of a PIT entry reflects the name of requested content, and the second – a set of interfaces from which interests for this content have arrived.

When an NDN router receives an interest, it first looks up its PIT to determine whether an interest for the same named content is currently outstanding. There are three possible outcomes:

1. If the same name is already in the router's PIT and the arrival interface of the present interest is already in the set of *arrival-interfaces* of the corresponding PIT entry, the interest is discarded.
2. If a PIT entry for the same name exists and the arrival interface is new, the router updates the PIT entry by adding a new interface to the set; the interest is not forwarded further.
3. Otherwise, the router creates a new PIT entry and forwards the present interest using its FIB.

Upon receipt of the interest, the producer injects content into the network, thus *satisfying* the interest. The requested content is then forwarded towards the consumer, traversing – in reverse – the path of the corresponding interest. Each router on the path flushes state (deletes the PIT entry) containing the satisfied interest and forwards the content on all arrival interfaces of the associated PIT entry. In addition, each router (optionally) caches a copy of forwarded content in its local Content Store (CS). Unlike their IP counterparts, NDN routers can forward interests out on multiple interfaces in order to maximize the chances of quickly retrieving requested content.

The above description of interest forwarding only applies to content that has not been recently requested, i.e., not present in CS-s of intervening routers. Whereas, a router that receives an interest for already-cached content does not forward the interest further; it simply returns cached content and retains no state about the interest.

3. MOTIVATION

As follows from the previous discussion, the NDN architecture is primarily geared to applications that disseminate content. Routers directly assist content distribution by, whenever possible, satisfying consumer interests with cached content. This is different from IP, since NDN decouples the flow of content from the notion of content location.

In this paper, we focus on bi-directional conversation-style applications over NDN. Representative applications of this class are: audio/video conferencing, interactive chat, and remote login.

To support this communication paradigm in NDN, each end-point (e.g., Alice and Bob) must register its own namespace, and during the conversation, play the role of both producer and consumer of content. Before describing the session establishment and data exchange protocol, we refer to Figure 1 for a high-level overview. To initiate a session, Alice issues an interest for Bob's namespace, embedding her own namespace prefix in the name (as a suffix). Bob receives the interest and parses the name which indicates that Alice is requesting a session. Bob then responds with a content acknowledging agreement to establish a session. Thereafter, data flows in both directions: Alice and Bob exchange interests for each other's namespaces and generate content accordingly. Such a session can be viewed as two flows: Alice and Bob each request (and receive) the other party's content via interests. NDN routers that forward interests and content between Alice and Bob are oblivious to the conversation. The two flows (Alice→Bob and Bob→Alice) might even wind up using asymmetric paths. This type of communication does not get the main benefit of NDN router-side caching, since content is only intended to be received by one end-point.

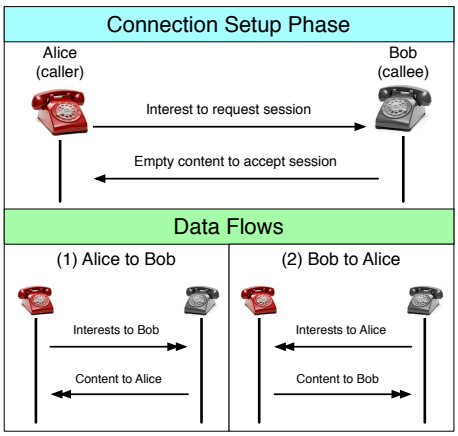


Figure 1: Bi-directional communication between Alice and Bob.

We note that NDN, in its current state, is not well-suited for this type of communication. Even though, as the above description shows, bi-directional communication can be supported by NDN, the result is somewhat awkward and inefficient. Consider what it takes for Alice and Bob to exchange one content packet in each direction: (1) Alice issues an interest in Bob's next content, (2) Bob replies with requested content, (3) Bob issues an interest in Alice's next content, and, finally, (4) Alice replies. All this requires two interests and two content packets. Each of these packets traverses a sequence of NDN routers and requires separate processing. For each router, an interest packet entails [9]:

(i.1) packet reception from layer below
(i.2) PIT look-up (existing entry for the same name?)
(i.3) creation of a new PIT entry
(i.4) FIB look-up, and
(i.5) forwarding to next hop

Note that step $i.2$ is designed to collapse duplicate interests, i.e., those issued by different consumers for the same content. However, in our setting of a point-to-point session, (i.2) is nearly useless, since only Alice issues interests for Bob's content and vice-versa. Its only remaining utility is in handling inadvertent retransmissions. Also, given that numerous interests flow from Alice to Bob as part of the same session, performing (i.4) for each interest seems wasteful.[1]

[1]Routing could conceivably change while a session is in progress;

Furthermore, for each content packet in either direction, every intervening router must perform:

(c.1) packet reception from layer below
(c.2) PIT look-up: content name→pending interest
(c.3) caching content in CS
(c.4) deletion of a PIT entry, and
(c.5) forwarding to next hop, where the interest came from

In summary, to exchange a content packet in each direction, each router must perform (i.1)-(i.5) and (c.1)-(c.5) operations twice. We believe that the corresponding overall amount of "work" is excessive and can be optimized for better performance. More generally, we believe that making NDN more friendly to bi-directional point-to-point communication is worthwhile, since, as discussed earlier, applications that involve this type of communication are here to stay, and their requirements are quite distinct from those of content distribution.

We note that router caching benefits not only efficient content distribiution. It also facilitates efficient recovery from packet loss, i.e., if a content packet is lost (e.g., due to wireless channel error or mobility), a consumer can re-issue an interest obtain desired content from some nearby router's cache. NDN modificications proposed in the next section retains this feature.

4. DESIGN

One of the key modifications we proposed is adding a new (third) packet type to the NDN architecture, which we refer to as a *bundle packet*, geared specifically for conversation-style applications. A bundle packet essentially combines a content packet with an interest packet, both of which travel to the same end-point. Intuitively, the main idea is as follows:

> Suppose that Bob receives an interest for its next content (e.g., keystroke or voice frame, depending on the application) from Alice. Suppose that Bob also needs to obtain Alice's next content. Instead of responding to Alice's interest with a content packet and separately issuing an interest for Alice's next content, Bob bundles the latter with the former.

Using bundle packets offers two benefits:

1. Fewer packets means that there are fewer API invocations, i.e., steps (i.1)/(c.1) and (i.5)/(c.5) are conjoined.
2. No FIB look-up (i.4) needs to be performed for a bundled interest since it travels along with content for which router PIT state already indicates the next hop.

In more detail, processing a bundle packet by an NDN router involves the following actions:

(p.1) packet reception from the layer below
(p.2) PIT look-up: content name→pending interest
(p.3) caching bundled content in CS
(p.4) create new PIT entry for bundled interest
(p.5) deletion of the original PIT entry, and
(p.6) forwarding bundle packet to the next hop

Compared with steps (c.1)-(c.5) and (i.1)-(i.5), processing a bundle packet is much more efficient. In fact, the only extra action performed by a router over and above processing a separate content packet is (p.4) – creation of a new PIT entry, while avoiding all processing steps associated with an interest packet. At the same time, sending a bundle packet is functionally equivalent to sending an interest and a content packets separately.

we discuss this in Section 4.2.

The rest of this section describes the construction and processing of bundle packets.

4.1 Constructing Bundle Packets

We consider two ways of forming bundle packets: concatenation and embedding. The most obvious way to form a bundle packet is to simply concatenate an interest packet to a content packet while leaving them essentially intact, except for a flag in the content header indicating that this is a bundle packet.

The second approach is similar, however, the interest is now embedded within the content packet. In this case, additional fields are needed to specify the interest offset. The main functional distinction of this approach is that the entire packet (both content and interest parts) are covered by the producer's (sender's) signature. This has certain security implications, which we discuss below.

There are some obvious trade-offs between concatenation and embedding. The former allows more flexibility, since any router that processes a bundle packet can easily decouple interest and content and forward them separately. This allows routers to apply local policy and choose whether to treat bundle packets as one unit or un-bundle them. It also allows bundle support to be introduced incrementally. For example, an NDN router that knows that its next-hop neighbors do not support bundling must decouple bundle packets before forwarding.

In case of embedding, this flexibility is lost since the producer's signature covers the entire bundle packet and decoupling is impossible without violating one of the main NDN tenets – verifiability of content packet signatures by any NDN entity, including routers. (Clearly, a router can neither recompute nor modify the content producer's signature). On the other hand, embedding offers better overall security, since interests can be authenticated along with content. This helps in mitigating so-called *interest flooding* attacks [10].

Another potential consideration is privacy: NDN interests are not signed by design since a public key signature leaks its source, i.e., the signer's (content producer's) identity. In case of embedding, interests are signed along with content. However, privacy of NDN interests is most relevant in the context of distribution of popular content, i.e., situations where multiple consumers request the same content. This is very different from our setting of point-to-point bi-directional communication where both end-points are well aware of each other. We also consider the privacy issue from the perspective of NDN routers. Normally, an NDN router does not learn the identity of the source of an interest (content consumer). It only learns the identity of the producer of content that satisfies that interest. However, in the context of processing a bundle packet, an NDN router learns that the content and the bundled interest are originated by the same entity.

We also note that, in terms of privacy, there is almost no difference between concatenation and embedding. Regardless of whether or not a bundled interest is covered by a signature, any entity that sees a bundle packet clearly learns the origin of both the content and the interest components.

While some loss of privacy seems to be inherent to the use of bundle packets, there are also potential avenues for security and privacy improvements per directional flow. For example, routers can apply certain policies to protect conversational flows from eavesdroppers. They can discard interests received off-path, thus preventing any additional parties from accessing session content. This might, however, impact error recovery mechanisms, e.g., if a route change causes an end-point to re-issue an interest over a different path.

4.2 Using Bundle Packets

As mentioned earlier, applications that involve continuous bi-directional (point-to-point) communication stand to benefit the most from bundle packets. Such applications generate interest and content packets in a synchronous manner, with data continuously flowing in both directions. Audio/video conferencing is one example of this application class. Such applications tend to have strict timing constraints on bandwidth and latency in order to ensure satisfactory end-user experience. For example, we consider an audio conferencing application running over plain NDN – without bundling. We then show how bundling helps and discuss some issues related to packet loss.

1. Alice and Bob each launch their audio-conferencing application, registering their respective namespace to receive interests and publish content (voice data): `/abc/alice/voice` and `/xyz/bob/voice`.
2. Alice initiates a call to Bob by issuing an interest for: `/xyz/bob/call/[alice]`. The `[alice]` component is an opaque embedding of Alice namespace prefix so Bob knows where to send subsequent interests.
3. Bob upon receipt of the interest, parses the suffix component to determine Alice's namespace. Bob accepts the call responding with a content packet as an acknowledgement.
4. Bob sends an interest anticipating content from Alice. The first such interest is: `/abc/alice/call/[bob]/0`. The trailing component ("0") represents the initial sequence number of Alice's content which Bob wants to retrieve.
5. Alice responds with the initial content and also issues an interest expecting content from Bob: `/xyz/bob/call/[alice]/0`.

Each party generates content at some negotiated rate, e.g., every 20ms (i.e., $rate = 50 pkts/sec$). Interests are issued with increasing sequence numbers to keep pace with available content. Typically a *sliding window* mechanism is used to achieve pipelining. The window size w (corresponding to the maximum number of outstanding interests), is selected such that the streaming of content overlaps with the round-trip time (RTT), i.e., $w = \lceil \mathrm{RTT} \cdot rate \rceil$. This ensures that both parties receive a continuous audio stream throughout the session.

In Steps 3 and 4, Bob can start taking advantage of bundle packets by combining the acknowledgement and his first interest. For her part, upon receipt of Bob's first bundle packet, Alice can respond with a bundle packet with her own audio content and an interest requesting audio content from Bob. Hereafter, both parties continue to exchange bundle packets in lock-step, until the end of the session.

An important advantage of using bundle packets is that each intervening NDN router performs **only one FIB look-up** for the entire session. (With a sliding window, w route lookups are performed.) This single FIB look-up is done in Step 1, when Alice issues her initial interest to Bob. This represents substantial savings for NDN routers, resulting in lower overall latency. Also, due to using bundle packets in a lock-step fashon, Alice and Bob use the same route in both directions; this route is "fixed" by Alice's initial interest. (Albeit, with a sliding window, up to w routes might be used.) This provides a reliable measure for RTT between the two end-points, as determined by Alice upon receipt of the first bundle packet.

The above scenario and expected savings occur under ideal network conditions, with stable routing and low congestion. Dynamic route changes and packet loss would certainly cause disruption. To recover from such events, the application can (and should) temporarily revert to sending interest and content packets separately. Bundling can be resumed once a new reliable path between the parties is established.

The above scenario shows that conferencing applications are well-matched to bundle packets. However, we emphasize that this technique is equally applicable to any applications requiring low-latency bi-directional communication. For example, a file transfer application can combine interests for additional content with acknowledgement packets. Also, an interactive chat application can bundle keep-alive (content) and interest packets if there is no actual data ready to be sent, to prevent PIT entries created by bundles from expiring.

5. IMPLEMENTATION

Our implementation of bundle packet support is based on the open-source NDN prototype, CCNx [11], which was originally developed at the Palo Alto Research Center (PARC) and was later made available to the academic and research community.

The main components of CCNx are the software forwarder (`ccnd`) and the CCN client library (`libccn`). Both are implemented in C. Packets in CCNx have a free-form structure and allow for variable sized fields. The prototype does not impose any limitation on the field size or packet length. Packets are encoded using a compact binary XML representation known as CCNx Binary Encoding ("ccnb") [12], which is designed specifically for CCNx. The client library provides full support for encoding and decoding ccnb-structured data and provides high-level API functions to create ccnb-encoded interest and content packets.

The `ccnd` forwarder implements the NDN router functionality, including FIB, PIT, and Content Store (CS). All packets are sent and received through a "face". A face extends the notion of a network interface to include virtual transport mechanisms such as TCP/UDP tunnels and inter-process communication mechanisms. In addition to interfacing with other routers, this allows applications to communicate directly through a face. In order to support bundle packets, we: (1) define a new packet structure; (2) specify new encoding and parsing functions in CCNx client library; and (3) modify the forwarder to support the new packet type. Our implementation is based on CCNx 0.5.1.

As discussed in Section 2, we consider two approaches – concatenation and embedding – to forming a bundle packet. Although they can coexist, we currently only provide support for the former. The encode function takes a ccnb-encoded interest and content as input and returns a ccnb-encoded bundle packet. The packet structure defines a distinguishing tag labelled `Bundle` to allow parsers to distinguish it from `Interest` and `Content` tags. The rest of the layout is followed by two tagged arbitrary-size binary objects containing the corresponding content and interest.

A bundle packet parser function is also included. It takes a bundle packet and extracts encapsulated interest and content packets returning a copy of both as ccnb-encoded data.

An application sends a bundle packet in the same manner as an interest. If a bundle packet is issued, the interest packet is extracted, registered in the client PIT along with a callback handler to process the corresponding incoming content. Then, the bundle packet is delivered over the IPC socket connecting to the forwarder.

The `ccnd` forwarder runs an event loop which does the following: (1) polls each face for incoming data and assembles ccnb-encoded packets, (2) processes packets according to their type, and (3) forwards any outstanding data. Code that handles (1) and (3) is agnostic with respect to data being sent or received. Therefore, we only needed to modify (2) to include support for bundle packets.

The packet processing function parses a ccnb-encoded data stream and determines whether the opening tag is `Interest` or `Content`. It then calls the appropriate handler. We add a branch to distinguish a `Bundle` tag, and call a custom bundle handler function.

The bundle handler extracts the encapsulated interest and content components. First, a modified interest handler function is called, duplicating the functionality of the original interest handler, except for propagating the interest. The forwarding look-up is retained in case the content does not satisfy any interests. The modified content handler is called with both the content and bundle messages. The content is then processed in the same manner as in the original handler, except the bundle packet is cached in the CS. If an interest is satisfied, the bundle packet is placed into the outbound queue of the outgoing face.

6. EXPERIMENTS

We built a prototype file exchange (bi-directional transfer) application between two users. This application comes in two flavors: with and without bundle support. To perform a fair comparison, the prototype with bundle support is run exclusively on the modified CCNx code base. The other prototype runs on the original unmodified CCNx code. We refer to the two application instances running on the users' hosts as A (Alice) and B (Bob).

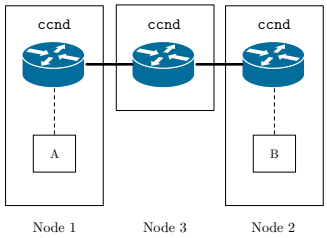


Figure 2: Topology used in our experiments. Node 3 is the forwarder, Nodes 1 and 2 run applications A (Alice) and B (Bob), respectively.

Topology used in our experiments is composed of three NDN nodes. The first node runs A and a copy of the `ccnd` process that acts as a local NDN router. Similarly, the second node runs B and `ccnd`. Finally, the third node (*forwarder*) forwards traffic from A and B via its local copy of `ccnd`.

Mutual file transfer is started by A, which sends an interest to B signaling the beginning of the process. B responds with the corresponding content and with its own interests for A's content.

The two parties send a total of $10,000$ interests each, retrieving the same number of content packets. Payload size of each content packet is $1,000$ bytes. The total amount of data sent from A to B (and, from B to A) is about 10MBytes. We also performed experiments with content packets with payload sizes, varying between 100 and $4,000$ bytes. We omit these results due to space constraints, since they are virtually identical to those presented below.

Application A runs on a host with a quad-core Intel Xeon E5620 2.4GHz processors with 12GB of memory. The forwarder runs on an machine with two quad-core Intel Xeon E5420 at 2.5GHz equipped with 16GB of RAM. Application B runs on a node equipped with an Intel Core2Duo 2.13GHz processor and 3GB memory. All machines run Ubuntu Linux and are connected using 100Mbps full-duplex Ethernet links with a maximum MTU of 1500 bytes.

In order to achieve maximum throughput our prototype uses pipelining, via sliding window, as described in 4.2. In particular, A sends w interest packets to B. For every interest from A, B replies with the appropriate content packet followed by an interest for A's content. Similarly, for each interest received from B, A replies with a content packet and a new interest. After that, A and B continue the exchange in lock-step until the transfer completes.

When using bundle packets with a sliding window, the first w interests from A and the last w content packets received by B are not bundled. (In other words, they are sent as regular interest and content packets, respectively). All other packets are bundled. For example, if $w = 1$, A sends one interest and B responds with a bundle packet. A issues its next bundle packet upon receiving one from B. This continues until the very last bundle from A to B. Upon receipt of that last bundle, A has no further interests and it sends the remaining content packet to B.

Clearly, w is impacts packet processing by the forwarder. A large w might cause several packets to be queued in the forwarder's buffer, introducing delay. Since the choice of w significantly affects performance, we experimented with varying w between 1 and 40. For each experiment, we measured the following:

1. **Forwarding Processing Time (FPT)**: time for the forwarder to (logically) forward a content packet *and* and an interest packet in either direction. With bundling, we measured FPT as the time to forward a single bundle packet. Without it, FPT is measured as the sum of the times needed to separately forward a content packet and an interest packet.
2. **Round Trip Time (RTT)**: time for an interest to retrieve its corresponding content packet. With bundle packets, we measured RTT as the time for a bundle packet to retrieve the corresponding content (contained within a bundle) from the other party. This slightly penalizes the bundle approach, since measured RTT corresponds to the transfer of a larger amount of data than in the non-bundle case.
3. **Transfer Time (TT)**: time to transfer all data involved in the experiments (10 MBytes of content) from A to B.

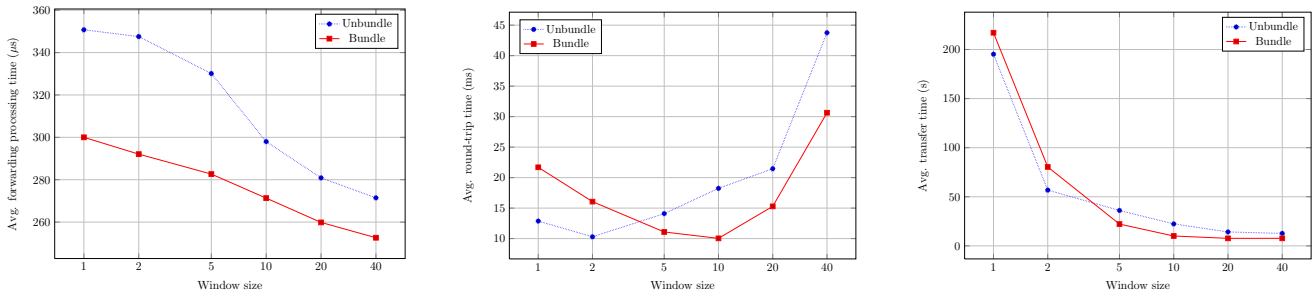
6.1 Experimental Results

Figure 3a illustrates FPT for variable window sizes. Results show that, for all window sizes considered, FPT of a bundle packet is lower than that of an interest plus a content packet. In particular, for $w = 1$, average processing time decreases by 14.5% for the bundle case. Meanwhile, for $w = 2$, bundling provides the biggest improvement – processing time decreases by 16%. For larger window sizes, the difference between the two approaches gets smaller. However, it remains significant with a minimum of 6.9% for $w = 40$. This confirms that, by reducing the number of FIB lookups and lowering layer 2-3 and API call overhead for processing interest and content packets separately, the bundle approach allows the forwarder to achieve significantly better performance.

Note that there is a decreasing trend in FPT as we increase window size for both cases. This is partially because of the way CCNx handles packets in its incoming buffer: as we increase window size, the number of invocations of packet (interest or content) processing routines in `ccnd` decreases. In other words, when multiple packets are (temporarily) stored in a router's incoming buffer, `ccnd` can pull and process many of them at once.

Figure 3b shows average RTT for our experiments. In it, for $w < 5$, RTT of non-bundled packets is lower. The reason is that bundled packets are larger than single interest or content packets. In the non-bundled case, the smallest RTT is achieved when $w = 2$. This reflects our observation above: when `ccnd`'s incoming buffer contains multiple packets, such packets are processed together, thus saving forwarding time. When $w = 1$, the forwarder's buffer always contains at most one packet; with $w = 2$, the buffer often contains two packets. However, further increasing w does not provide additional benefits for non-bundled case, since savings in processing time are outweighed by the waiting time of multiple packets in the forwarder's buffer.

Since bundle packets have lower processing overhead compared to interest/content packet pairs, the smallest RTT occurs when



(a) (b) (c)

Figure 3: (a) Forwarding Processing Time (FPT) (b) Round Trip Time (RTT) and (c) Transfer Time (TT) for various window sizes.

$w = 10$. This RTT is almost identical to that of non-bundled case with $w = 2$. As shown below, larger window size implies better throughput. Therefore bundling allows our test application to achieve higher throughput with minimum latency.

Figure 3c shows the transfer time (TT) for both bundle and non-bundle cases for various window sizes. For small w, TT is higher for bundle packets. This accounts for the fact that, in our setup, the optimal window size is greater than 2: in this case, RTT becomes a limiting factor for the transfer rate, since the end-points end up waiting rather than issuing new interest and content packets.

As expected, TT decreases for large values of w, where $w \geq 5$, thus improving the transfer rate. An application should ideally scale its window size, with an initial setting to at least the Bandwidth-Delay Product [13] of the session.

7. RELATED WORK

There has been no previous work in terms of combining requests and responses in bi-directional communication over CCN. Somewhat related work has been done for TCP with selective acknowledgments (SACK) [14] and its extension, duplicate-SACK (D-SACK) [15]. SACK and D-SACK allow the receiver to selectively acknowledge correctly received packets, such that the sender must only re-send the packets that have been lost. This is a significant improvement over older cumulative TCP ack techniques in presence of multiple packet loss from one window of data.

In [16], Clark et al. studies the effect of bi-directional traffic on TCP congestion control algorithm. In particular, it was observed that, in the BSD Tahoe TCP implementation, packets from a single connection are clustered together, similarly to what was earlier observed in [17] for one-way traffic. This causes ACK compression, which significantly reduces available bandwidth for TCP connections. [16] also shows that, for two-way traffic, the issue of ACK compression is made worse by t he interaction of ACKs and data packets in the queue.

8. CONCLUSION AND FUTURE WORK

In this paper we show that NDN, in its current state, is not well-suited for bi-directional low-latency point-to-point communication. To provide better efficiency for applications requiring this type of communication, we introduce a new bundle packet type that bundles a content with an interest traveling in the same direction. This results is roughly half the number of overall packets and reduces processing time in NDN routers, which translates into improved end-to-end throughput and lower round-trip time. Furthermore, introduction of bundle packets preserves existing NDN architectural features. We conducted extensive experiments demonstrating substantial performance due to the use of bundle packets.

Clearly, this work is only intended as a first step towards efficient and reliable bi-directional point-to-point communication over NDN. Items for future work include the following:

- Additional experiments over more complex topologies, including the official NDN testbed [18], in ordr to determine the impact of reduced processing time over longer routes.
- Extending the bundle model to handle bi-directional multicast traffic, such as audio/video conferencing among multiple users or sites.
- Evaluating the impact of many concurrent flows (rather than just a single one) between two or more nodes.

Furthermore, we plan to identify, and experiment with, other classes of traffic that can benefit from relatively small changes to the NDN architecture.

9. REFERENCES

[1] "Internet Protocol - DARPA Inernet Program, Protocol Specification," RFC 791 (Proposed Standard), Sep. 1981.

[2] "Named data networking project (NDN)," http://named-data.org.

[3] M. Gritter and D. Cheriton, "An architecture for content routing support in the internet," in *USENIX USITS*, 2001.

[4] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," in *ACM CoNEXT*, 2009.

[5] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM*, vol. 37, 2007, pp. 181–192.

[6] V. Jacobson, D. Smetters, N. Briggs, M. Plass, J. Thornton, and R. Braynard, "VoCCN: Voice-over content centric networks," in *ReArch*, 2009.

[7] Z. Zhu, J. Burke, L. Zhang, P. Gasti, Y. Lu, and V. Jacobson, "A new approach to securing audio conference tools," in *AINTEC*, 2011.

[8] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, "ANDaNA: Anonymous named data networking application," in *NDSS*, 2012.

[9] www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html, retrieved Jul. 2012.

[10] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS & DDoS in Named-Data Networking," UC Irvine, Tech. Rep., 2012.

[11] "Content centric networking (CCNx) project," http://www.ccnx.org.

[12] www.ccnx.org/releases/ccnx-0.5.1/doc/technical/BinaryEncoding.html, retrieved Jul. 2012.

[13] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323 (Proposed Standard), May 1992.

[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018 (Proposed Standard), Oct. 1996.

[15] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," RFC 2883, Jul. 2000.

[16] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *ACM SIGCOMM CCR*, 1991, pp. 133–147.

[17] S. Shenker, L. Zhang, and D. Clark, "Some observations on the dynamics of a congestion control algorithm," 1990.

[18] "Official NDN testbed," http://www.named-data.net/testbed.html.