Communication Patterns for Web Interaction in Named Data Networking

Ilya Moiseenko
UCLA
iliamo@cs.ucla.edu

Mark Stapp
Cisco Systems
mjs@cisco.com

David Oran
Cisco Systems
oran@cisco.com

ABSTRACT

Named Data Networking (NDN) is an information-centric networking architecture that has recently attracted significant attention. At first glance NDN's pure pull-based communication model seems to match the request-reply mechanics of HTTP/Web interactions. In reality, modern Web communication patterns involve passing client-side information and/or application state in requests. As we attempt to apply these communication patterns to NDN, we find that it is not immediately clear how to use NDN effectively. In this paper, we examine multiple diverse approaches to running modern Web-like applications over the NDN communication architecture, discussing advantages and drawbacks of each of the proposed approaches. Our primary goal is to start a focused discussion of how NDN can support modern Web communication patterns effectively.

Categories and Subject Descriptors

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

Keywords

NDN; REST; Web

1. INTRODUCTION

The Web today is a universal platform for many kinds of services, from familiar content browsing and media streaming to purpose-built applications hosted in browsers and in stand-alone agents. The backbone of the web is the HTTP protocol [1] [2], which is based on a request/response model running on top of a point-to-point connection to a server. A client sends a request in the form of a message containing a URI [3], request meta-information, and possible body content. The server responds with a message containing entity meta-information, and possible entity-body content.

Named Data Networking is a recently proposed general-purpose, information-centric network architecture [4] [5]. It uses a pull-based model: clients send requests into the network in order to retrieve data; no other unsolicited transmission is allowed. NDN defines two types of network packets; the two possess highly asymmetric properties. Clients send Interest packets, which contain only a name and a minimal set of additional control fields. Servers respond with Data packets, which contain the data associated with the name in the corresponding Interest.

In this paper, we examine diverse approaches to matching the needs of an important category of modern applications to the capabilities of the NDN protocol architecture. Challenges present themselves when we attempt to support Web communication as it might take place over a future NDN internet. Our goal is to stimulate progress on this topic in the research community. We do not believe it is either desirable or straightforward to simply reproduce HTTP bit-for-bit within an NDN protocol envelope. Rather, we explore the approach of enabling NDN-based Web clients to pass the necessary meta-information and application state to Web server applications that use NDN. We start with the necessary abstract communication patterns and use those patterns to explore how they might be realized within NDN protocol mechanics.

In Section 2 we provide a brief overview of the NDN architecture. In Section 3 we describe the current state-of-the-art with Web applications over the IP Internet and explain some of the immediate problems that arise when IP is replaced with NDN. In Section 4 we introduce NDN communication patterns suitable for Web-like interaction and provide a high-level examination of their advantages and drawbacks. Section 5 describes an analytic model used to quantify the efficiency of the proposed NDN communication patterns. Section 6 contains a summary of our observations and a comparison chart of the communication patterns discussed in this paper.

2. NAMED DATA NETWORKING

Named Data Networking (NDN) proposes the replacement of IP's endpoint-to-endpoint communication model with one centered around named objects. NDN offers two distinct packet types: *Interest* and *Data*. Both types carry a *name*, which uniquely identifies an information object. Names in NDN are hierarchically structured, and contain distinct components. Large objects that cannot be carried in a single Data packet are *segmented* into multiple packets; the segment number is carried as a component at the end of the name. An example representation of an NDN name might be: `/com/example/data/object/1`. Applications are expected to design appropriate naming schemes for the kinds of communication they require.

To retrieve data, a consumer requests it by sending an Interest packet containing the name of the desired content. A router uses this name and its Forwarding Information Base (FIB) to forward the Interest closer to the location of the data. When an Interest reaches an entity who has a matching Data packet, the Data packet is returned to the consumer. Each router who forwards the Interest

ICN'14, September 24–26, 2014, Paris, France.
Copyright 2014 ACM 978-1-4503-3206-4/14/09 ...$15.00.
http://dx.doi.org/10.1145/2660129.2660152.

constructs an entry in a Pending Interest Table (PIT). Each PIT entry contains the Interest name, the ingress interface(s) where the Interest has arrived, and the egress interface(s) to which the Interest has been forwarded. Data packets are returned by following the reverse path of the Interest, using the per-Interest state kept in the PIT.

NDN routers can cache any passing Data packet in a data structure called a Content Store (CS). A cached Data packet can be used to satisfy an Interest. One of NDN's central tenets is that a consumer does not care whether a Data packet was served from a router cache or from the original producer: the trust in data is decoupled from the place and time the data was obtained, and any consumer can validate the integrity and provenance of the content using the Data packet's signature.

Crucially, one Interest packet returns at most one Data packet. A client retrieves a large, segmented object by sending multiple Interests naming each segment. In NDN, there are no unsolicited Data packets; the PIT mechanism in a router prevents unsolicited Data packets from being processed or forwarded. This symmetry between Interest and Data packets is often called *flow balance*.

3. WEB INTERACTION

Arguably, the most widespread and economically important of the Internet's application infrastructures is the modern Web, which uses HTTP/TCP/IP as its foundational protocol underpinnings. In addition to carrying content data, the HTTP protocol defines a wide range of meta-data for both requests and responses. The meta-data are carried in HTTP headers, part of the messaging protocol. The meta-data sent in client requests may be information about the client application itself, including information about acceptable content languages and data encodings.

A large fraction of these Web applications use a transactional paradigm known as Representational State Transfer (REST) [6]. REST improves scalability by distributing application state from servers to clients. A pure RESTful request is self-contained — it carries all the information necessary for a service to process the request. Without the client-side context, a RESTful service may be inefficient or impaired, or may not be able to function at all. The familiar HTTP cookie is a simple form of distributed context, where a service uses the HTTP protocol to convey tokens, often opaque, to its clients. The tokens are typically unique to each client; this allows the service to associate multiple requests from a given client together. The cookie may carry client-side state directly, or may be used as a reference to state held at the server.

The usability of these distributed applications depends on the latency between user action and the rendering of a result. Because the Web is composed of multiple highly distributed services, this issue of latency (round-trips within the transport or the application protocol) has considerable importance in the system design. Modern browsers and other applications have evolved to be ever more efficient in the way they use round-trips, by caching content locally, and by reusing DNS information and TCP connections. Some modern browsers even speculatively initiate DNS queries and TCP connection activity in order to improve perceived responsiveness [7].

As we view the current state of HTTP/RESTful communication, then, we see these key points:

- Clients have data to send in their requests, in the form of HTTP header meta-data and other application-specific RESTful state.
- Many client requests are intimately bound to the client context data associated with them; the context and the request are carried together in the HTTP communication protocol messages. The client-specific data tends to make each request unique even when clients are accessing common resources or services.
- Latency and number of network round-trips are key factors in efficiency and perceived responsiveness.

Given our understanding of the existing NDN protocols, RESTful interactions encounter a number of challenges:

- All of the client-side context and meta-data associated with a request must be encoded in the Interest name field: no other field is present in the base NDN architecture.
- NDN Content objects are immutable, and the object names are bound to fixed data. Services are not able to return different results based on client-specific processing unless the clients use unique names in their requests.
- NDN's stateful forwarding supports clients (consumers) who do not have to have a globally-routable address (name). Web services that require bidirectional data flow cannot get their own requests to their clients unless the clients have routable names.
- Many web sites and RESTful applications depend on being able to identify (or at least count) the specific clients making requests. NDN mechanisms like Interest aggregation and pervasive caching prevent producers from seeing some Interest packets.

In the next section, we will examine a range of approaches to supporting RESTful and Web applications on NDN networks. Each approach addresses the challenges we have outlined above in a different way, adapting the basic NDN protocols in more or less significant ways.

4. NDN COMMUNICATION PATTERNS

In this section, we explore communication patterns suitable for running transactional and interactive REST- or Web-like applications over NDN. We focus on the key issue as we see it: how can servers obtain the client meta-data and context information that is associated with client requests? We start with an approach that relies solely on the existing design of NDN, but several important drawbacks compel us to try out alternative patterns which introduce various degrees of changes in NDN. The two types of NDN packets divide the discussion into two corresponding categories: approaches using clients' Interest packets alone, and approaches where the server is obliged to retrieve information from client-sourced Data packets.

We discuss benefits and disadvantages of each communication pattern as well as the effect each has on the network, considering several specific factors as we examine each approach:

- Interest name size: impact on routers
- Data name size: impact on Data packet efficiency
- Round-trips
- Client data segmentation and reliable delivery
- Potential security vulnerabilities: reflection, amplification, flooding, spoofing/poisoning

4.1 Name Component

In the basic NDN protocol design, an Interest packet carries little more than a Name field. In some previous work, Interest names have been used to pass commands to an NDN router [8], to pass authenticated requests to a lighting controller [9], and to convey the current state of a system to support distributed dataset synchronization [10].

Today's interactive Web applications need to pass meta-information and application-specific data with their requests, so we begin by examining the consequences of using the Interest's Name field to convey that information. An Interest packet is routed through the network via the name it carries. Application meta-information and client-side data required for a particular type of request could be carried by appending it using one or more trailing name components. This pattern is illustrated in Figure 1.

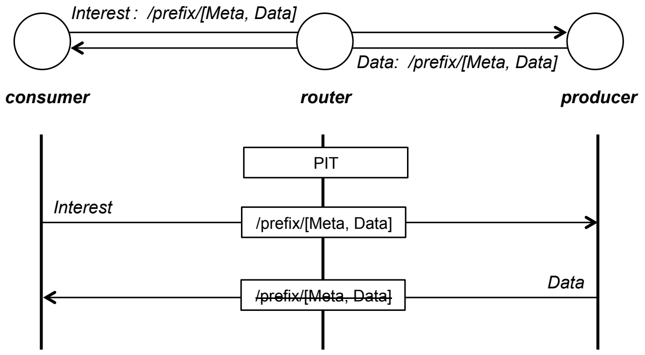


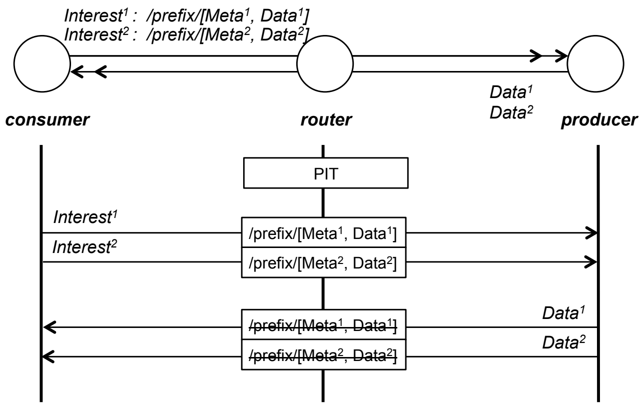
Figure 1: Interest name carries client-side information.

This communication pattern works with the current NDN architecture, and naively seems fairly natural. The client-side data is bound to each Interest packet directly, satisfying the server's expectation that the client-side context will be present along with each client request.

However, there are a number of significant drawbacks to this simple approach. The first concern is related to stateful packet forwarding in NDN. Contemporary HTTP requests that perform browsing often convey hundreds of bytes (or even kilobytes) of supplemental information in HTTP headers [11] [12]. If this meta-information and application-specific data is placed in the Interest name, there may be a significant additional overhead on intermediary NDN routers. Each router will have to process these large names, increasing the computational load, and the accumulated name state held in their PIT data structures will consume substantially more memory.

A second concern is decreased network throughput and increased nodal processing delays. The entire name must be echoed in each Data packet. Inside the NDN router, longer names may lead to more operations on name components, slowing down packet processing. The name-to-payload ratio can turn out to be far from optimal. Regardless of the eventual fragmentation scheme NDN proposes, large names will reduce available packet space, reducing space for the actual content. This leads to decreased goodput, and potentially more fragmentation and reassembly operations per Data packet.

A third concern is the possibility of cases where a single Interest name is not able to carry all required application data. While there is no clear consensus within the NDN community on the maximum allowed size of the name, there is a clear possibility that meta-information and application data (e.g. HTTP POST or a large cookie) may be larger than the maximum name length can accommodate. Within the constraints of the current NDN protocol, meta-information and application data would have to be subdivided into multiple Interests' names, transmitted as multiple Interest packets and reassembled by the producer. Figure 2 illustrates how NDN might accommodate transmitting arbitrary sized client-side data to the producer, and retrieving an arbitrary sized response from it.



Interest^1 : /prefix/[Meta^1, Data^1]
Interest^2 : /prefix/[Meta^2, Data^2]
Data^1
Data^2
consumer
router
producer
PIT
Interest^1
Interest^2
/prefix/[Meta^1, Data^1]
/prefix/[Meta^2, Data^2]
~~/prefix/[Meta^1, Data^1]~~
~~/prefix/[Meta^2, Data^2]~~
Data^1
Data^2

Figure 2: Client data carried in multiple Interests.

According to this pattern, the consumer sends no fewer Interests than needed to both accommodate client-side data in Interests and fetch all segments of the producer's reply. This pattern appears to take only a single round-trip to transmit the whole request and receive the whole reply. But once multiple related packet transmissions are introduced, we now need to consider some sort of reliable delivery of consumer-supplied information. That is, the client must re-transmit its Interests in the absence of any timely response or acknowledgement that they have been delivered. This complexity leads us to examine some alternative protocol approaches.

The conventional design of NDN interactions is that the producer acknowledges arrival of Interest packets in its Data packets, but the situation may be more nuanced. The completion time for Web and application requests requiring dynamic on-demand content can vary widely. As a result, it is not clear how the client should estimate waiting time between Interest retransmissions. One extreme is to use an Interest retransmission timer at the scale of network RTT. But this may result in many unnecessary retransmissions of Interests if the server processing time is significantly greater than RTT. The other extreme is to use a timer scaled to the tolerable application response delay. This in turn results in poor responsiveness in cases when network retransmission is indeed necessary. NDN protocol mechanics do not inherently distinguish network-level and application-level responsiveness, despite the substantially differing time scales.

One solution might be for the producer to use Data packets to acknowledge delivery of Interest packets containing meta-information and application data. The producer application would acknowledge each of these Interest packets prior to the execution of the actual content request, resulting in a two-phase operation. First, an initial set of Interest packets conveys the client-side data; Data packets from the producer acknowledge receipt of this information. Then a second round of Interest packets retrieves the actual producer-side content. This pattern is illustrated in Figure 3.

Note that this approach employs parallel Interest transmission to reduce overall latency. Separating the delivery of client data from the Interests used to retrieve producer content eliminates the need for all Interests in the exchange to use the same name, reducing the Interest size penalty. However, a significant problem with this approach is that the first round of 'acknowledgement' Data packets must be signed with the producer's private key in order to be considered valid. Signing a Data packet is computationally costly. If malicious clients flood Interests like these, this could lead to a denial of service (DoS) attack on the producer. In addition, the producer requires some means of associating the client-side data in

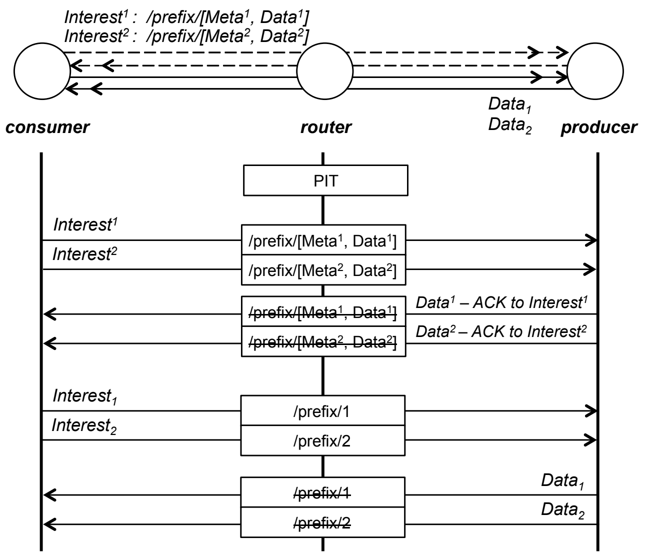

Interest^1 : /prefix/[Meta^1, Data^1]
Interest^2 : /prefix/[Meta^2, Data^2]
Data$_1$
Data$_2$
consumer
router
producer
PIT
Interest^1
Interest^2
/prefix/[Meta^1, Data^1]
/prefix/[Meta^2, Data^2]
~~/prefix/[Meta^1, Data^1]~~
~~/prefix/[Meta^2, Data^2]~~
Data^1 – ACK to Interest^1
Data^2 – ACK to Interest^2
Interest$_1$
Interest$_2$
/prefix/1
/prefix/2
~~/prefix/1~~
~~/prefix/2~~
Data$_1$
Data$_2$

Figure 3: Two-phase Interest exchange.

the initial round of Interests with the subsequent Interests for the producer's content, resulting in more interaction state at the server.

4.2 Compressed name component

Including significant client-side data in Interest names raises concerns about memory scalability for the PITs of intermediary NDN routers, and decreased throughput due to the need to echo the entire name in each Data packet. These concerns can be partially addressed by compressing the client-side data into a constant size compact representation, and using this representation in the router PIT and in Data messages.

To achieve compression, a specialized Name component could be introduced to hold client meta-information and application data. An NDN router recognizing this specialized name component could then compute a hash of the component. This operation would effectively reduce the amount of state held in the PIT, compressing variable meta-information and application data into a constant size hash value. In order to forward Data packets back to the consumer, the producer application would replace the specialized name component with the corresponding hash value. As a result, Data packet names would continue to match the names in the PITs of intermediary routers, while occupying less space. This technique is illustrated in Figure 4.

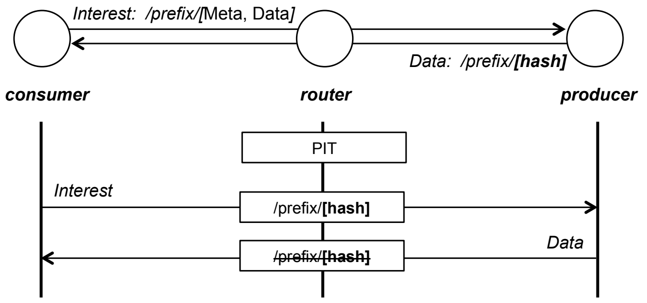


Figure 4: Consumer-supplied name-component is compressed to a hash.

4.3 Common Issues with Interest Names

Even with name component compression, all protocol approaches where meta-information and application data are pushed in Interest packet name components still have a number of common problems.

Exposure of meta-information and application data impairs confidentiality. If meta-information similar to HTTP cookies and HTTP headers such as Referer and User-agent are passed unencrypted in an Interest name component, the user can be easily tracked and deanonymized by a third-party observer. If security-sensitive data is held in these meta-information structures, the compromise could be even more substantial.

Signature generation must be performed on-the-fly for all Data packets that acknowledge the arrival of Interest packets with names carrying meta-information and application data. The per-client information creates names that are unpredictable, so the producer application must build and sign the corresponding Data packets dynamically. This introduces a potential vulnerability to a resource-exhaustion attack. NDN signature generation with public key cryptography is computationally expensive — significantly more expensive than, for example, SYN cookie generation.

Interest packet flooding in NDN networks can be a vector for Distributed Denial of Service (DDoS) attacks [13]. It has been shown that many Interest flooding attacks can be mitigated by exploiting stateful forwarding in NDN routers, such as by observing the rate with which Interests successfully retrieve Data packets on a per-prefix per-interface basis [14]. If meta-information and application data is pushed in Interests and if producer applications acknowledge every Interest with a Data packet, the per-prefix per-interface statistics may be distorted. An artificially high Interest satisfaction rate might jeopardize detection and mitigation of Interest flooding attacks.

4.4 Application Data field

We have examined some approaches to carrying client-side information in Interest names; now we'll explore sending request meta-data in the Interest packet, but outside the Interest name. In this approach, an Interest carries the additional application data in an *ApplicationData* field. This field would contain opaque data, and thereby not influence the operation of NDN routers or their processing in any way. The Interest name only requests the named content, and does not carry any client- or application-specific information. Figure 5 illustrates this approach.

The client includes an AppData field in its "base" Interest packet - the Interest for segment zero of a possibly segmented Data object. For Web-like interactions, the AppData field would carry meta-information about the client application, including stored cookies (i.e. what is found today in HTTP headers). In a standalone RESTful application, the field would carry client-side application context data.

The AppData field is opaque to routers. No special name components are present, and no special name processing takes place at routers. If a client Interest packets name a cacheable object, intermediate routers can perform normal CS processing and return the cached data. If an application requires server-side processing, client Interests must use unique-ified names so that Interests from different clients avoid aggregation.

The client does not have to send the entire AppData in each Interest during a multi-segment exchange. In an ongoing exchange of packets to retrieve larger, segmented Data objects, the server may need to associate the correct client context with each individual Interest in order to respond properly. To accomplish this, the

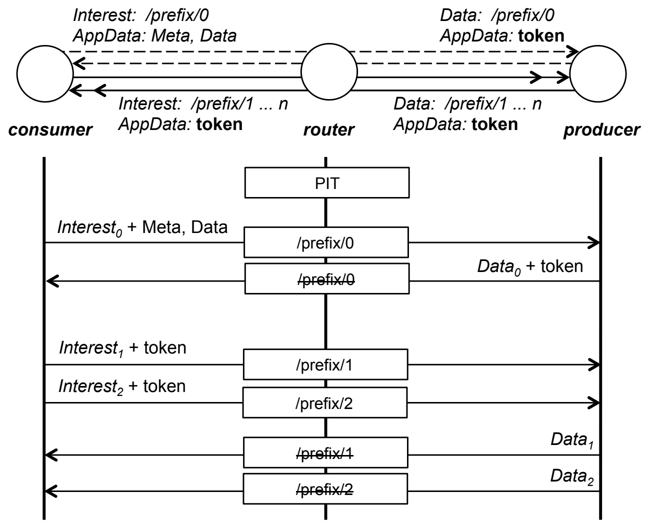


Figure 5: Interest carrying ApplicationData field.

server could generate a token — presumably shorter than the entire client context data — and return it to the client with the first Data packet. Subsequent Interests then would include this token in the AppData, allowing the server to properly associate the client metadata with each individual Interest packet. If a series of exchanges required dynamic, frequently updated client context, obviously that context would have to be transferred between client and server as it changed.

Employing such a token mechanism requires that each Interest contain either the client context, or a corresponding server-generated token. This may affect a client's choice of initial Interest window size. If the initial Interest window size is just one, data fetching efficiency during the first round trip is reduced. If the initial Interest window is greater than one, the client has not been offered a server-side token, so it must transmit redundant application data in each Interest in this window. The choice of initial window size for Interests using a scheme like this may have delicate trade-offs.

The communication pattern with the Application Data field has the following benefits:

- The Interest name does not need any special processing. There is no need for complex name matching at the PIT or CS: exact-match for names is available.
- The application context information travels directly with the Interests; the client context, name, and returning data remain bound together.
- The application data can be transferred just once, with the initial Interest. Subsequent Interests can refer to the context if a server-generated token is returned in Data packets.
- No additional round-trips are needed.

Any scheme that "pushes" client data in Interest packets increases Interest packet size, possibly substantially. The NDN property of flow balance assumes that Interest packets will generally be small compared to the corresponding Data packets. Pushing 'unsolicited' data might compromise that property. To address this concern we might consider a limit on the size of Interest packets. A 4KB limit, for example, would be adequate for most current Web-like interactions [11]. However, this is still quite large — possibly large enough to make bandwidth accounting for Interests more important. A RESTful application that required a larger client payload would need to send multiple Interests, or use a different mechanism.

4.5 Data Locator field

The alternative to pushing client-side data with Interest packets is a communication pattern where the producer application pulls data it needs from the client. An essential piece of such protocols is a so called Interest-Interest exchange [15]. In this exchange, an initial Interest packet is expressed by the consumer application as usual. This initial Interest prompts the producer application to express one or more Interest packets in return. These requests from the producer retrieve client-specific information from the client; the producer then uses that information to satisfy the client's original Interests.

The Interest-Interest information could be placed in the initial client Interest name, but this approach would suffer from some of the same constraints as the examples in the previous sections — extremely long NDN names have drawbacks. Enclosing one name in another, for example, will not allow both names to approach their maximum lengths, which is inconvenient for application designers.

In our view, a better alternative would be to introduce an optional *DataLocator* field in the Interest packet. The presence of the DataLocator would serve as an indication for the producer that some supplemental information — meta-information, consumer-supplied data, etc. — is available to be fetched from the client before processing the initial request. The DataLocator would therefore contain a name the producer could use to express Interests that reach the client application. We discuss some variations of this mechanism below.

4.5.1 Routable name

This pattern requires the consumer application to provide a routable name at which it can be reached. The client must be prepared to package necessary meta information and application data in properly-formatted and signed Data packet(s). The consumer application might acquire a routable prefix from the point of presence (PoP) of the Internet Service Provider (ISP) that it is currently connected to, or through some other means.

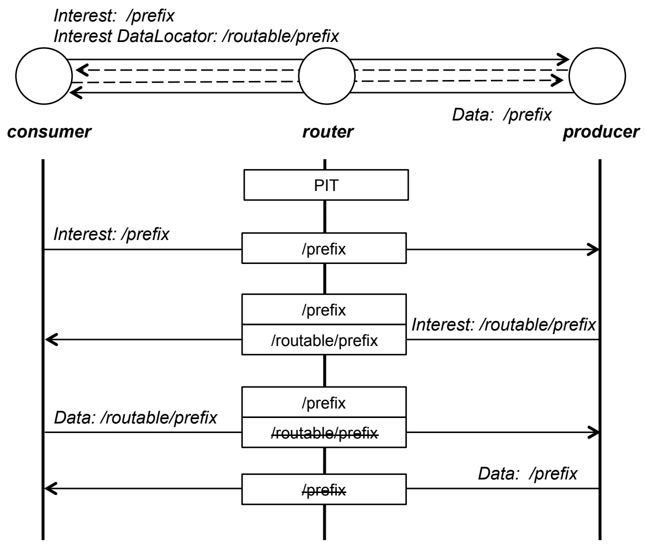


Figure 6: Interest-Interest exchange with routable name.

The consumer application sends an Interest packet containing the name for the producer to use in a DataLocator field. When the producer application receives the Interest, it transmits an Interest packet using the name specified in the DataLocator field to fetch meta-information and/or application data associated with the client's request. This communication pattern is illustrated in Figure 6.

The immediate advantage of this protocol is eliminating "pushed" data from client's Interests, which do not need to convey more than a single name. This restores the NDN flow balance property. A second important benefit is that the producer application is now in control of the data retrieval process. The producer is subject to standard NDN flow control and congestion control mechanisms as it retrieves Data from the client.

A third benefit is that some client-side data can benefit from NDN's natural on-path caching. Web cookies that represent the state of the server, kept on the client, may be stable for extended periods of time. Client data associated with related idempotent requests (e.g. HTTP GETs) can be cached in the intermediary routers that are located closer to the producer. Both the client and server therefore benefit from the NDN mechanisms that localize traffic and reduce latency.

However, the use of routable names for the server to fetch client data has several drawbacks. First, the client must acquire and convey a routable name prefix. A mobile consumer will either have to acquire a new prefix every time its connectivity changes, or use some sort of indirection service to map a stable name alias to its current routable prefix. This adds complexity, and introduces the possibility of traffic interruptions.

Second, the DataLocator mechanism's use of a routable name could be used to launch a reflection attack involving the producer. If an attacker specifies the name of a target third party, the producer will be induced to direct Interests to that third party. The reflection attack might be mitigated if the DataLocator is inspected when Interests enter the client's Internet Service Provider (ISP) network. The ISP ingress router could perform a check similar to an ingress filter in Reverse Path Forwarding (RPF) [16], accepting and forwarding Interest packets carrying DataLocators that will route to the source face. The router would drop any Interests with DataLocator names that would route elsewhere.

4.5.2 *Non-routable transient name*

The problems caused by the use of routable prefixes in the DataLocator field prompt us to explore the possibility of using non-routable prefixes for client-side data. This approach uses the per-packet router PIT state to construct an ephemeral path for Interests going back from the producer to the client in a manner somewhat like Kite [17]. As shown in (Figure 7), this introduces several changes in the forwarding mechanism of an NDN router:

1. The client constructs a unique name, preferably using a distinguished (by convention) non-routable prefix, and includes it in a DataLocator field.
2. When an Interest containing a DataLocator field arrives at a router, the DataLocator name is saved in the PIT along with the name in the Interest packet itself.
3. The producer responds with an Interest using the non-routable name taken from the DataLocator. As the producer's Interest moves through the network, each NDN router performs an exact match on the producer's Interest name using the extended PIT entries created as it forwarded the client's original Interest. If the router finds a match, it creates a new PIT entry for the non-routable name with the egress interface matching the ingress interface of the original Interest. The FIB is not consulted: the producer's Interest is forwarded on the inverse path of the consumer's original Interest packet using the PIT alone.

The DataLocator name is not independently routable. If the server (or anyone else) tries to access this information object out-

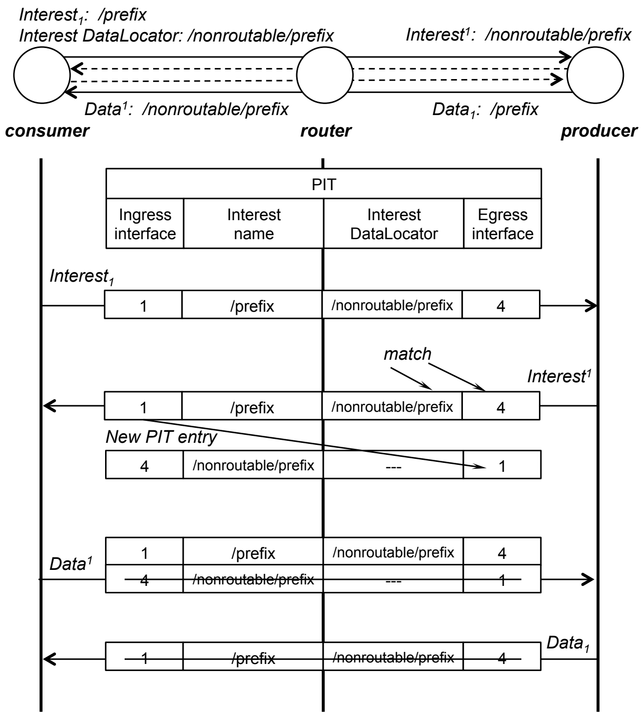


Figure 7: Interest-Interest exchange with non-routable name.

side the context of the enclosing Interest/Data exchange, the operation will fail. Further, since the names used cannot be forwarded outside the reverse path, reflection attacks are eliminated.

The fact that these non-routable Interests bypass the normal FIB does not prevent them from being satisfied by a Content Store. If a router's CS cache has a matching entry, this entry can be returned to the producer. However, the non-routable name can take any form, including self-certifying and other flat names, and therefore reverse forwarding cannot depend on longest prefix lookup.

When a mobile consumer changes its connectivity, the path for reverse Interest packets can be quickly rebuilt by client-side retransmission of its Interest packet, which will create necessary PIT state again.

If client meta information or application data is too large to fit in one Data packet, the consumer application segments it into multiple Data packets just as would be done for any large Data object. The producer application issues multiple interests to retrieve the entire information object. In order to accommodate this, the algorithm matching DataLocator names in the PIT ignores any segment number name component. Pipelining would allow a producer to fetch arbitrary size client data with minimal round trips.

5. ANALYTIC MODEL

In this section, we develop a simple analytic model and use it to characterize each communication pattern. We model bidirectional traffic between an HTTP/Web-like client (consumer) and an HTTP/Web-like server (producer); network traffic is an obvious, key metric applicable to all of the communication patterns we have considered.

The model applies NDN *segmentation* as data objects grow large. Segmentation is the operation where a content producer splits a large data object into smaller pieces, naming and signing each separately. NDN flow balance, the one-to-one correspondence between Interest and Data packets, assumes that Data packets have constant

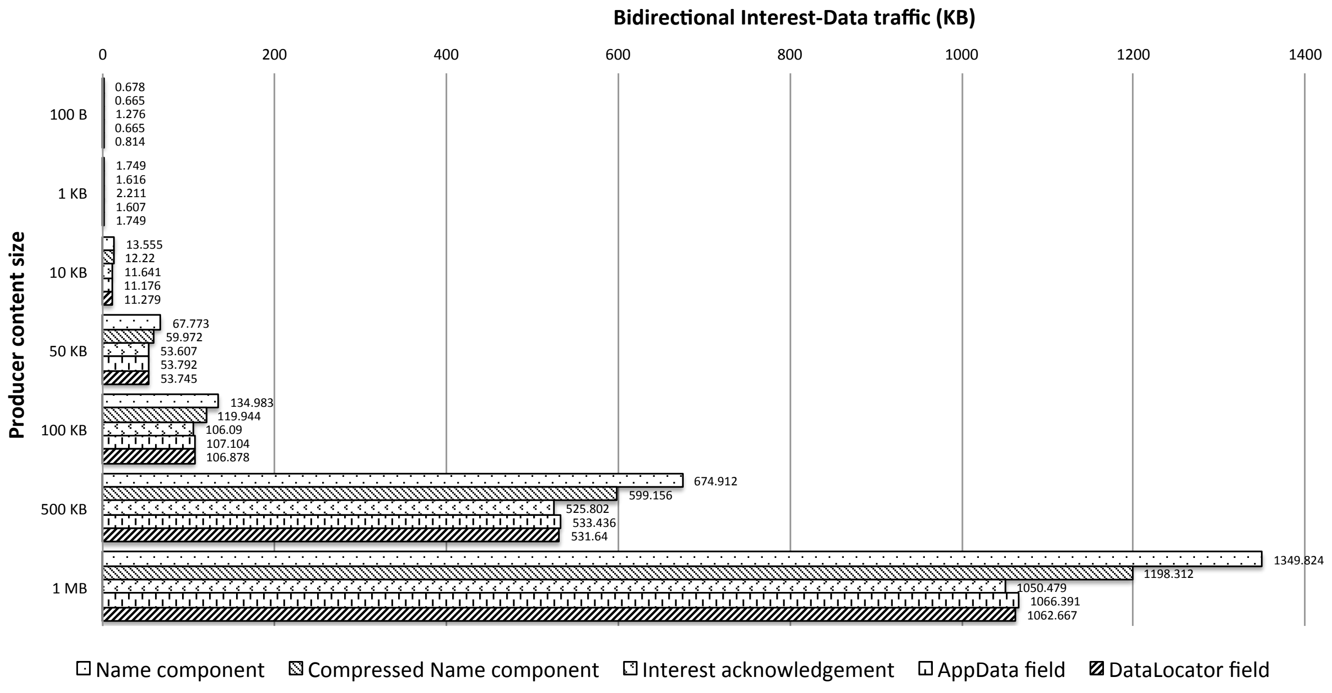


Figure 8: Bidirectional traffic model using 512 bytes of client data.

size for simplifying hop-by-hop flow- and congestion control. A large name field reduces the available space for the content payload in each fixed-size Data packet. A given content object requires more or fewer segments (packets) depending on the payload space made available in each pattern.

The model utilizes a base Interest name (prefix) that is 50 bytes long, with 512 bytes of client-side data. We do not argue that this is accurate or representative of actual traffic; rather that it is not unrealistic given the current web traffic patterns [11].

We use this simplified arithmetic equation to compute the number of segments (*NoS*):

$$Number\ of\ segments\ (NoS) = \frac{Producer\ content\ size}{Space\ for\ content}$$

The choice of the communication pattern affects the amount of space available for content in Data packets. In the name component pattern, all of the client data is appended to the base name prefix. In the compressed name component pattern, Data packet names have a large hash value (e.g. SHA-512) appended to the base name prefix. In the Interest acknowledgement pattern, client data is not echoed in the name of Data segments. In the application data pattern, producer generates and echoes back a token (e.g. SHA-256), carried in its Data segments. In the DataLocator pattern, client data is not echoed in the name of Data segments.

$$Space = Data\ size - \begin{cases} Prefix - Client\ data & (Name\ component) \\ Prefix - Hash & (Compressed\ name) \\ Prefix & (Interest\ ack.) \\ Prefix - Token & (Application\ data) \\ Prefix & (Data\ Locator) \end{cases}$$

Name component pattern carries client data in the Interest name. Each segment is fetched with an Interest carrying the relatively large name.

$$Interest\ traffic = NoS * (Prefix + Client\ data)$$

Large names force the producer to send more segments, increasing the amount of bidirectional traffic:

$$2way\ traffic = Interest\ traffic + NoS * Data\ size$$

Compressed name component pattern carries client data in each Interest name, but echoes back only the hash of the client data in each Data packet of the producer's response.

$$Interest\ traffic = NoS * (Prefix + Client\ data)$$

Total amount of bidirectional traffic:

$$2way\ traffic = Interest\ traffic + NoS * Data\ size$$

Interest acknowledgement pattern uses an initial series of Interests containing client data, acknowledged with signed Data packets. The producer's actual content is then fetched using the normal length Interest packets.

First round — acknowledged delivery of client data. A Data packet with Interest acknowledgement has a negligible payload, therefore:

$$1st\ round\ traffic = (Prefix + Client\ Data) * 2$$

Second round — fetching segmented content from the producer.

$$2nd\ round\ traffic = NoS * prefix + NoS * Data\ size$$

Total amount of bidirectional traffic:

$$2way\ traffic = 1st\ round\ traffic + 2nd\ round\ traffic$$

Application Data pattern carries client data in a single Interest, in a special Interest packet field. The producer generates and echoes back a token, carried in its Data segments. Subsequent Interests, if any, use the producer's token and do not have to convey the client data explicitly.

$$Interesttraffic = (Prefix{+}Clientdata){+}NoS{*}(Prefix{+}Token)$$

Total amount of bidirectional traffic:

$$2way\ traffic = Interest\ traffic + NoS * Data\ size$$

Pattern / **Criteria**	Name component	Compressed Name component	Interest acknowledgement	Application Data field	Routable DataLocator	Non-routable DataLocator
Impact on router memory	**Large**	Normal	**Large**	Normal	Normal	2 x Normal
Payload/Name ratio in Data packets	**Low**	Normal	**Low**	Normal	Normal	Normal
Round trips	1 round	1 round	**2 rounds**	1 round	**2 rounds**	**2 rounds**
Support of large client data	**No**	**No**	Multiple Interest packets	**No**	Multiple Data packets	Multiple Data packets
Retransmission of client data	**Slow, timescale of application RTT**	**Slow, timescale of application RTT**	Fast, timescale of network RTT	Fast, timescale of network RTT	Fast, timescale of network RTT	Fast, timescale of network RTT
Disruption scenarios	PIT inflation DoS on router's fragmentation & reassembly	DoS on router's hashing	DoS on producer's signing DDoS with Interest flooding		Client mobility Reflection attack	

Table 1: Comparison chart of communication patterns.

DataLocator pattern carries an additional name in a special field of each Interest packet. For the purposes of this arithmetic traffic model, the *routable* and *non-routable* locator names are identical. We use 50-byte name lengths for both the content name and the DataLocator name.

The client's Interest packets carry the content name and a DataLocator name. The producer responds with its own Interest(s) using the DataLocator name.

$$Interest\text{-}Interest\ traffic = 2 * Prefix * NoS + Prefix$$

The client provides its client data in Data object(s) using the DataLocator name, then retrieves the actual content from the producer.

$$Data\text{-}Data\ traffic = Data\ size + NoS * Data\ size$$

Total amount of bidirectional traffic.

$$2way\ traffic = Interest\text{-}Interest\ traffic + Data\text{-}Data\ traffic$$

Figure 8 summarizes the results of applying this arithmetic model. The canonical NDN Interest formulation proves to be noticibly less efficient than any other. Name field size has an obvious impact for any but the smallest contents; even the use of a compressed name component has a considerable though less-dramatic impact. The network bandwidth used by the other three protocol patterns is roughly equivalent. This particular metric does not distinguish among these other approaches particularly, though we highlight some key comparison points in Table 1 and discuss it in the next section.

6. CONCLUSION

In this paper we have explored the characteristics of HTTP or REST-like interactions using the NDN communication model. Active state transfer from the client to the server is one of the defining characteristics of these interactions. This state transfer from client (consumer) to server (producer) in NDN is challenging, for a number of reasons:

- Interest packets are lightweight: they do not have a "payload" field to carry ancillary parameters, and have no confidentiality protection mechanisms without additional protocol machinery.
- Interests create state on every NDN router, whose size is correlated with the size of the names carried.
- Interests are aggregated in order to reduce the router state and save upstream bandwidth. Additional name manipulation needs to be implemented in clients to guarantee Interest propagation all the way to the producer application where this is necessary.
- NDN clients (consumers) are not required to have a routable name, making it difficult for them to 'publish' data as NDN content objects.

We describe a number of communication patterns to enable NDN to handle RESTful transactions. All, in our view, have significant drawbacks or disadvantages, as summarized in Table 1.

The **Name component** pattern has an impact on all processing and message efficiency, due to name size expansion.

The **Compressed name component** pattern violates the NDN assumption that Interests and returning Data packets use identical names. Interests still carry potentially large names; there is no way to perform fast retransmission of Interest packets without additional protocol machinery.

The **Interest acknowledgement** pattern uses two phases: one set of Interests conveys client-side data, and a second retrieves the producer-side content. This helps by allowing retransmission of client data without waiting for a server timeout, but imposes a burden on producers. It also may frustrate Interest flooding mitigation techniques by skewing the network statistics they depend on.

The **Application Data field** pattern is efficient, but requires changes in the Interest packet format, and potentially challenges the inherent NDN assumption that Interest packets are small.

The **Routable DataLocator** pattern not only requires changes in the Interest packet format, but also depends on a client's obtaining a routable prefix to be able to respond to Interests sent by the producer. This pattern is awkward for mobile clients, and may make NDN services vulnerable to reflection attacks.

The **Non-routable DataLocator** pattern exploits stateful forwarding to instantiate a transient reverse path for Interests sent by the producer. Since no information in the FIB is being used for reverse forwarding, the DataLocator name can be non-routable. The consumer does not need to obtain a routable prefix, but the router's forwarding algorithm is more complicated.

Eventually, these Web-like/REST-like interactions in NDN may evolve in ways that make them significantly different from their current forms in the IP Internet. But some fundamental constraints or properties will continue to hold: technical feasibility (e.g. scalability, latency) that affect all distributed, REST-ful services, and business needs that drive the 'commercial' Web. In our opinion, REST-like applications are important, and therefore it is highly desirable that they be supported effectively. This deserves consideration in the ongoing discussion and design of the NDN architecture.

7. REFERENCES

[1] T. Berners-Lee, R. Fielding, and H. Frystyk, "RFC 1945: Hypertext Transfer Protocol–HTTP/1.0," *The Internet Society*, 1996.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol–HTTP/1.1," *The Internet Society*, 1999.

[3] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 3986: Uniform resource identifier (URI): Generic syntax," *The Internet Society*, 2005.

[4] 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.

[5] L. Zhang et al., "Named data networking (NDN) project," NDN Project, Tech. Rep. NDN-0001, October 2010.

[6] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM TOIT*, vol. 2, no. 2, pp. 115–150, 2002.

[7] I. Grigorik. High performance networking in google chrome. [Online]. Available: http://www.igvita.com/posa/high-performance-networking-in-google-chrome/

[8] CCNX documentation. [Online]. Available: https://www.ccnx.org/releases/latest/doc/technical/Registration.html

[9] J. Burke, A. Horn, and A. Marianantoni, "Authenticated lighting control using Named Data Networking," *UCLA, NDN Technical Report NDN-0011*, 2012.

[10] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *ICNP*, 2013.

[11] B. Newton, K. Jeffay, and J. Aikat, "The Continued Evolution of Web Traffic," in *MASCOTS*. IEEE Computer Society, 2013, pp. 80–89.

[12] S. Ramachandran. Web metrics: Size and number of resources. [Online]. Available: https://developers.google.com/speed/articles/web-metrics

[13] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS and DDoS in Named Data Networking," in *ICCCN 2013*. IEEE, 2013, pp. 1–7.

[14] 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.

[15] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control," *arXiv preprint arXiv:1208.1336*, 2012.

[16] F. Baker and P. Savola, "RFC 3704: Ingress filtering for multihomed networks," Tech. Rep., 2004.

[17] Y. Zhang, H. Zhang, and L. Zhang, "Kite: A Mobility Support Scheme for NDN," NDN Project, Tech. Rep., 2014.