Connection-Free Reliable and Efficient Transport Services in the IP Internet

J.J. Garcia-Luna-Aceves and Abdulazaz Ali Albalawi
Computer Science and Engineering Department, University of California at Santa Cruz, CA 95064
jj@soe.ucsc.edu, aalbalaw@ucsc.edu

***Abstract*—The Internet Transport Protocol (ITP) is introduced to support reliable end-to-end transport services in the IP Internet without the need for end-to-end connections, changes to the Internet routing infrastructure, or modifications to name-resolution services. Results from simulation experiments show that ITP outperforms the Transmission Control Protocol (TCP) and the Named Data Networking (NDN) architecture, which requires replacing the Internet Protocol (IP). In addition, ITP allows transparent content caching while enforcing privacy.**

***Index Terms*—IP, TCP, NDN, transport protocols, connections**

I. Introduction

A transport protocol bridges the gap between the services available from a network infrastructure (e.g., the Internet) and the facilities desired by application processes using the network infrastructure. As such, the design of any transport protocol is about interprocess communication [24], [63], [67]. The Transmission Control Protocol (TCP) provides reliable transport services in the Internet today, and its design of TCP dates back to the ground-braking work by Cerf and Kahn on the *Transmission Control Program* [18]. Like most of the early work on reliable transport protocols [15], [63], [73], the design of the Transmission Control Program [19]–[21] evolved assuming the use of end-to-end connections [64] to implement reliable in-order delivery of data between remote processes. Since then, even after the original design by Cerf and Kahn [18] was divided into the Internet Protocol (IP) [50], [52] and TCP [51], the conventional wisdom has been that end-to-end connections between remote processes are *necessary* to provide reliable and efficient communication between remote processes [63] without involving the underlying communication infrastructure for anything other than best-effort delivery of datagrams.

Interestingly, Walden [68], one of the designers of the host-to-host protocols of the ARPANET, proposed a message-switching alternative to the connection-oriented ARPANET host-host protocol [15] based on his earlier work on a system for interprocess communication [67]. Walden's proposal was discussed by Cerf and Kahn [18], but connections became the norm for reliable communication between remote processes in the Internet, and message-oriented protocols like Walden's proposal have been viewed as inherently unreliable [64].

Many variants of TCP have been proposed over the years, and several transport protocols have been proposed to improve on its performance [26], [27], [35], [61]. However, all transport protocols designed to provide reliable communication have been based on end-to-end connections. To some extent, this prompted the recent development of several information-centric networking (ICN) architectures [3], [9], [71], which attempt to make Internet content delivery more efficient by eliminating connections but require major modifications to the network infrastructure. Section II provides a critique of important prior work related to reliable communication between remote processes over computer networks. What is most notable is that no prior connection-free transport protocol has been proposed that provides reliable communication over a datagram-based communication infrastructure.

This paper presents the **Internet Transport Protocol** (ITP) as an example of how connection-free reliable and efficient transport services can be provided over the IP Internet.

Section III describes the basic design of ITP, which integrates Walden's proposal on a message-switching host-host protocol [68], the receiver-initiated approach to reliable data transfer first described by Jacobson et al. [30], the use of manifests [40] advocated in several ICN architectures, and the inclusion of pointers to such manifests in each request for data and response.

Section IV summarizes the way ITP uses sockets and interacts with applications.

Section V describes how ITP enables transparent caching with privacy by encoding pieces of content in a way that only a process that obtains a manifest for a content or service is able to consume it.

Different approaches can be used for retransmission and flow control in ITP. However, Section VI summarizes retransmission and congestion control mechanisms for ITP that are very similar to those currently used in TCP (TCP Reno) to illustrate the inherent benefits of eliminating connections and using manifest pointers.

Section VII compares the performance of ITP, TCP, and Named-Data Networking (NDN) [43] using the ns3 [66] and ndnSIM [44] simulators. The results of the simulation experiments indicate that ITP is inherently more efficient than TCP and provides the same benefits of NDN without requiring any changes to IP, the Internet routing infrastructure, or the domain

This material is based upon work sponsored by the National Science Foundation (NSF) under Grant CCF-1733884. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or the U.S. government.

name system (DNS). Section VIII provides our conclusions and discusses promising avenues for future research.

II. Related Work

Considerable work has been reported over the years on transport protocols that provide reliable end-to-end communication [45], [49], as well as transport control strategies in ICN architectures [16].

Limited work has been reported on transport protocols that augment UDP with additional functionality (e.g., DCCP [34]) but do not guarantee reliable data delivery.

From its inception, TCP based its buffer management for retransmission and congestion-control using a byte stream as the abstraction [18]–[21], [51]. However, other transport protocols, like the CYCLADES end-to-end protocol [73], have used packets rather than bytes for the same purpose.

A plethora of proposals have been made to improve the performance or functionality of TCP [49]. The proposed improvements include more efficient flow control and retransmission strategies, the use of multiple connections, enabling multihoming of the remote processes exchanging data reliably [26], [27], [35], [61], and even the use of machine learning in TCP congestion control [2], [32], [70]. However, all this prior work assumes that reliable communication between remote processes requires end-to-end connections.

The correctness of algorithms for connection establishment and termination has been the subject of considerable work. Several results have been published [1], [37], [60], [69] showing that there is no connection-based protocol for reliable communication operating with a finite sequence-number space that works correctly under different conditions related to messages being deleted, duplicated, or reordered. In addition, it has been shown that no correct connection-based protocol exists that provides reliable communication in the presence of nodes failing and losing their state [37], [60]. Given these inherent limitations, the connection-based protocols used in practice today, including TCP, are based on a relaxed definition of correctness.

Existing connection-based protocols are based on connection initialization procedures similar to the *Five Packet Handshake* (FPH) [10]. FPH renders connection-based protocols that work in practice, provided that all messages are assigned unique identifiers. This implies that, even in the presence of node crashes, either no message identifier is ever generated more than once, or no message identifier is reused before every prior message using the same identifier has been eliminated from the system. In practice, FPH and similar connection-initialization approaches are implemented using a finite number of unique message identifiers assuming known bounds on message-delivery times, local processing times, and rate of new message creation.

Another important aspect regarding the evolution of transport protocols is their deployment [45]. Many novel solutions have not seen widespread deployment due to middleboxes (e.g., firewalls and NAT boxes) modifying or blocking any unfamiliar traffic [42], [55]. As a result, recent reliable transport protocols (e.g., QUIC [35]) adopt UDP for the naming of processes and framing of messages. Moreover, recent studies [17], [65] even suggest encapsulating TCP traffic inside UDP datagrams as it passes better through these middleboxes.

Transparent content caching is highly desirable to expedite content delivery on the Internet or ICN's [23]. However, current transparent-caching solutions exclude traffic carried over a secure closed connection, which is a problem because more than 50% of web traffic is served over HTTPS [25]. To overcome this issue, some proposals [36], [47] split a TLS session into two separate TLS sessions, one session between the consumer and the cache, on one side, and the cache and the producer on the other side. However, this requires caches to be trusted using a certification authority and introduces the problem that consumers cannot choose the level of security between the cache and the server. Other solutions [41], [59], suggest altering the TLS layer to support trusted proxies operations on traffic flows. Although these solutions support features such as intrusion-detection or content-filtering, they do not provide support for transparent caching.

NDN [43] eliminates end-to-end connections through modifications in the network infrastructure. However, NDN does not avoid the use of virtual connections altogether, because routers must maintain per-Interest state in their Pending Interest Tables (PIT) to forward data packets in response to Interests. The per-Interest state maintained by the routers between two communicating processes (a content consumer and a content provider) constitute a per-Interest virtual circuit, and the interplay between in-network caching and the added per-Interest state is still being studied [22]. Interestingly, the congestion-control schemes proposed for NDN and other ICN architectures to date are very similar to those used in TCP [3], [7], [12]–[14], [16].

To the best of our knowledge, Named-Data Transport (NDT) [5] is the the only prior approach that provides efficient and reliably content distribution without using end-to-end connections or changes to the Internet routing infrastructure. However, NDT requires modifications to the DNS.

III. ITP Design

The design of ITP uses the simple idea that the manifest of a named data object being exchanged between communicating processes can be used to provide the needed context for the exchange. The manifest of a data object is itself a data object and specifies: (a) The unique immutable name of the data object, (b) the structure of the data object consisting of object chunks (OC) that can be sent in messages, and (c) the procedure that should be used to decode the data object from a set of OC's. Additional metadata can be made part of the manifest of a data object, such as the names and size of object chunks and a list of IP addresses to contact to request them.

We denote by **nexus** the establishment of the context needed for an association between processes by means of manifests. Fig. 1 illustrates how a producer transmits a data object D to a consumer in ITP based on a nexus. ITP messages use

UDP headers stating the address and port of the consumer and producer; this is indicated in the figure with "UDP." The *manifest* of a data object D is denoted by $M(D)$.

The application running at the consumer site asks ITP to send a data packet, which requests a data object. The producer specifies a number of parameters through a specific system call in response to that request. The ITP process constructs a manifest and sends it to the client as a data packet called *Manifest Message* in Fig. 1, and creates a *Manifest Control Block* (MCB) that specifies $M(D)$, points to the memory location for D, and includes a *Producer Buffer* if additional memory space must be allocated for efficiency. The OC's of D are stored in a Content Store (CS) at the producer, and can be copied to the producer buffer to reduce latencies.

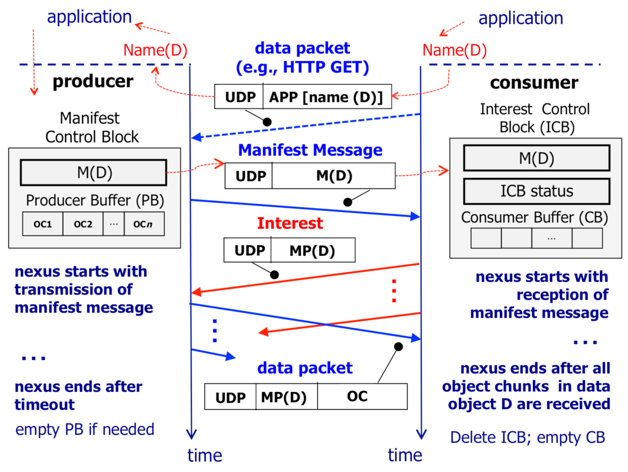

Fig. 1. The nexus in ITP replaces the end-to-end connection needed in TCP

The nexus for D at the producer starts when it creates the MCB for D and ends after a *nexus timeout* that must be long enough to allow consumer(s) to obtain the OC's in D, without having to reallocate memory for the Producer Buffer. The nexus timeout is restarted each time the producer sends a data packet with an OC of D.

The nexus for D at the consumer starts after it receives the manifest $M(D)$ and then creates an *Interest Control Block* (ICB) for D and allocates memory for a *Consumer Buffer* (CB) to store OCs of D. The ICB includes $M(D)$ and the ICB status indicating the OCs that have been received and those that are missing. The nexus at the consumer ends when it has all the OCs needed for D, at which point the ICB for D can be deleted. Once the consumer has a nexus for D, it obtains the data in D by sending requests that we call *Interests* for D. An Interest for D , denoted by $I(D)$ in Fig. 1, states: the names of the consumer and producer; the name of the data object; and a manifest pointer ($MP(D)$) that references $M(D)$ and states implicitly or explicitly the OCs that the consumer is missing and those that it has obtained. The producer responds to an Interest $I(D)$ with one or multiple data packets. Each data packet contains the manifest pointer $MP(D)$ of the Interest that prompted it and OC's that are part of D. The use of manifest pointers stated in Interests free the producer from having to maintain per-consumer state, and its nexus is simply with the data object D and its structure.

IV. Software Architecture and API

All entities in ITP have the same structure, making it a bi-directional messaging protocol. These entities can be viewed as a producer and a consumer.

An ITP producer is responsible for sending a data object, and an ITP consumer is responsible for consuming the object. When the server application sends out its reply, it is the ITP producer's responsibility to construct the Manifest for this data and send it to the ITP consumer at the other end. It is also its responsibility to send data packets in response to received Interests. The ITP consumer is responsible for retrieving the data using Interests based on the information provided by the manifest. The same occurs when the client application sends out a request (e.g., an HTTP GET). However, applications that consume data are naturally different from applications that produce data. Therefore, we specified different system calls to send the data over ITP that fits the application need. For example, when the client sends an HTTP GET request, it is done through a different system call than the one used by the server application to send an HTTP response. This system call triggers the ITP producer at the client side to deliver the Data packet encapsulating the HTTP GET request instead of constructing a Manifest and send it to the ITP consumer at the server end.

An ITP producer uses the MCB to remember the manifest itself, the manifest timeout, and consumers authorized to retrieve the data object. The MCP is similar to the transmission control block (TCB) used in TCP to maintain data about a connection. However, the MCB is only used to maintain consumer-independent state, because the ITP consumers are tasked with remembering nexus variables specific to them and state their values in the manifest pointers included in their Interests.

An ITP consumer maintains the *Interest Control Block* or ICB. A consumer creates an ICB for each new manifest it receives. The ICB includes variables such as manifest name, list of ITP producers to contact, Interest timeout and so forth. Once all the OC's of a data object are satisfied, the ICB tigers the ITP consumer to deliver the data object from the Content Store to the application.

An ITP producer can use OC's in the Content Store (CS) to satisfy Interests from different consumers, depending on the application need. A data object being retrieved is buffered in the CS until all its OC's are received and then it is delivered to the application by the ITP consumer. The decision of when to deliver the data to the application is issued by its ICB as mentioned before.

The interaction between transport protocols and applications can differ from one protocol to another. For example, the TCP API calls `bind()`, `listen()` and `accept()` are specific for server sockets and `connect()` is specific for client socket, while `send()` and `recv()` are common for both types.

Fig. 2 illustrates an example of a client/server relationship using the primary socket API functions and methods in ITP. The naming of these system calls is inspired by the early work by Walden on host-host protocols [67]. These calls are somewhat similar to the TCP/UDP socket API, but they differ significantly. The current TCP/IP socket API dates back to the 1980's with the release of what is called Berkeley sockets. Most of these socket calls prevent the programmers from understanding what goes on at the transport layer; instead, these calls just produce numeric error codes that usually have a generic meaning. Therefore, a future goal in the design of ITP API is to provide a platform that simplifies the programming of today's applications while hiding the complexity of the communication calls allowing application developers to customize their content distribution and have full control over deployment decisions.

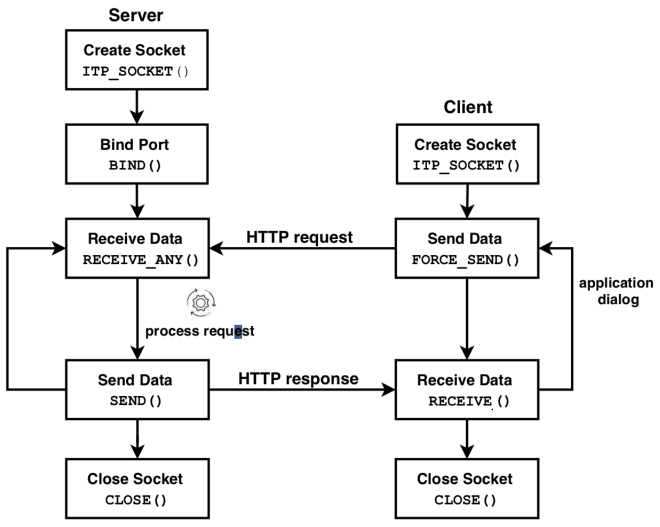

Fig. 2. Client and server communicating using ITP sockets

Given that no connection is established from the client to the server, the client just sends messages to the server using the `FORCE_SEND()` call. This system call forces the ITP producer at the client-side to send the message directly to the server without constructing a Manifest. Also, the server in ITP does not need to accept a connection, and instead, it just waits for messages to arrive. When a message arrives at the server, it contains the address (IP, Port) of the sender, which then the server can use to reply back to the client through the system call `SEND()`. As highlighted in Fig. 2, it is up to the **application dialog** to handle this. Because sockets by themselves are fully duplexed, an application can simply send back to the port of origin, as we mentioned before. An ITP server application can close its socket after the dialog ends; however, because there is no notion of a connection between the two ends, a server can simply close its socket.

V. Transparent Caching with Privacy

ITP allows arbitrary application traffic running over ITP to be cached at the transport layer, without the caching logic leaking to the application layer and without making any changes to IP. Network administrators can simply install a single ITP proxy cache in their network and configure a layer-four switch to redirect ITP traffic to ITP caches. Content carried over ITP can be made private and secure by adopting an approach like the one recently introduced for NDT [5].

Each data object in ITP carries a name, including the manifest. The name of the manifest is mapped to a specification of messages to be sent. A simple approach to name OC's in ITP is by using sequencing with the content name from the manifest. Using this method an ITP consumer appends a chunk number to the content name of the outgoing Interest and keeps incrementing it as needed, until it receives all the data packets with the OC's of the data object. The following sequence of ITP-Proxy steps for transparent caching of web traffic corresponds to the numbers shown in Figure 3.

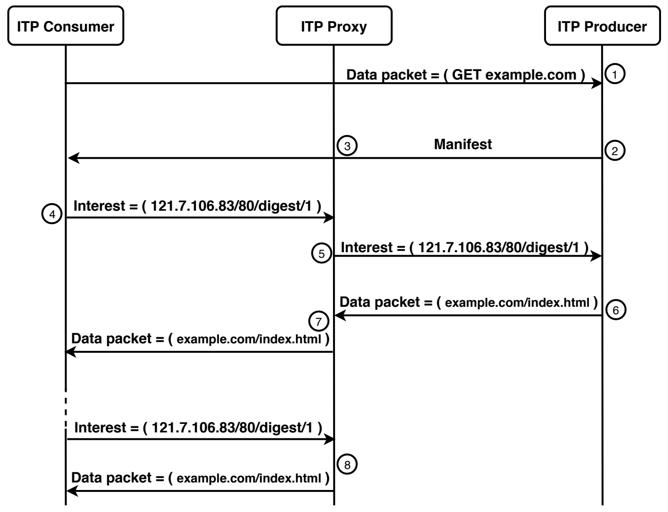
ITP Consumer
ITP Proxy
ITP Producer
Data packet = (GET example.com)
1
Manifest
3
2
Interest = (121.7.106.83/80/digest/1)
4
Interest = (121.7.106.83/80/digest/1)
5
Data packet = (example.com/index.html)
6
7
Data packet = (example.com/index.html)
Interest = (121.7.106.83/80/digest/1)
8
Data packet = (example.com/index.html)

Fig. 3. Transparent caching of ITP traffic

(1) A client sends an HTTP GET request for example.com to the web server. This request is sent over the system call `FORCE_SEND()`, which causes it to be sent as a data packet without the need for a manifest. The middle ITP Proxy on the way is configured to intercept Interests destined to a list of ITP producers and data packets from these ITP producers. It is up to the administrator to have cache data packets sent by the client applications to these web servers.
(2) After the ITP producer at the web server processes the incoming data packet it delivers it to the web server application along with the client IP address and port number. Once the application processes the message, it triggers a reply back to the client with the proper HTTP response using the system call `SEND()`. This causes the ITP producer to construct a manifest and send it to the ITP consumer at the client side.
(3) A layer-four switch intercepts the packet from the web server and redirects it to the ITP Proxy, which forwards it to the client.
(4) After processing the manifest, the ITP consumer sends an Interest to the ITP producer at the other end.
(5) The ITP Proxy on the way intercepts the Interest, then

checks whether it has the requested data packet in its content store (CS). Given that it does not, it forwards it to the ITP producer using the name in the Interest.

(6) After processing the Interest, the ITP producer responds with the requested data packet.

(7) The ITP Proxy intercepts the data packet and caches it in its CS if it does not exist to satisfy incoming future Interests. It then forwards it to the ITP consumer, which after processing it, will deliver it to the application layer.

(8) Finally, the Interests from a new ITP consumer (who has the same manifest) is satisfied at the ITP Proxy instead of going all the way to the ITP producer.

As the example shows, ITP caches do not keep track of pending Interests as it is done in NDN. As a result, forwarding Interests does not require any changes to the IP Internet routing infrastructure.

VI. Retransmission and Congestion Control

We summarize retransmission and congestion control strategies in ITP that are very similar to those in TCP to illustrate the inherent benefits of ITP compared to TCP. Our design takes into account three key differences between ITP and TCP. ITP is receiver-driven, because the use of a Manifest to describe a content object allows the ITP consumer to be in charge of controlling retransmissions and managing congestion by adjusting how it makes requests to the producer. ITP is connection-free by means of nexuses whose management is done with a message-switching approach. Lastly, data carried over ITP can be cached transparently at the transport layer.

ITP uses a consumer-driven selective repeat retransmission strategy in which the ITP consumer is in charge of the data flow from the ITP producer, which simply responds to Interests with data packets. The manifest pointer included in each Interest from the ITP consumer tells the ITP producer what portions of the content have been received and which ones are missing, which serves as a selective acknowledgment (ACK). The ITP consumer controls the flow of data traffic by controlling the sending rate of its Interests, and allows for a window of data packets to flow to the receiver in response to its Interests. The window size is adjusted based on the AIMD (Additive Increase Multiplicative Decrease) mechanism commonly used in TCP for the congestion window.

ITP uses simple Interest-based approach for congestion control in which one Interest from the consumer elicits one data packet from the producer. The ITP consumer maintains a congestion window (cwnd) that defines the maximum number of outstanding Interests allowed to send without receiving their data packets. Similar to TCP New Reno, the consumer in ITP increases its cwnd based on slow start [29], starting with transmitting one Interest and increasing the cwnd by one for each new received data packet. The slow start continues until a packet loss is detected, in that case the ITP consumer limits the Interest rate by reducing its cwnd accordingly.

The policy used in ITP to retransmit Interests resembles the one used for retransmissions in TCP Santa Cruz [46]. Each Interest carries a retransmission count that uniquely identifies the specific transmission of the Interest and hence each RTT measurement is accurate, just as in TCP Santa Cruz.

An ITP consumer retransmits a lost Interest once an out of order data packet is received based on the order in the transmitted list and after a time constraint is met. A lost Interest y initially transmitted at time t_i is retransmitted once the following constraint is met: As soon as a data packet arrives for any Interest transmitted at t_x where $(t_x > t_i)$, and $(t_{current} - t_i) >$ RTT, where $t_{current}$ is the current time and RTT is the the time it takes to send an Interest and receive the data packet for it. Once the ITP consumer detects a packet loss using fast retransmit, the ITP consumer reduces its congestion window by one half and set the the threshold to the new window size causing the consumer to go into congestion avoidance [29].

Given the control of Interest transmissions by the ITP consumer and the one-to-one correspondence between an Interest and a data packet, ITP does not need to rely on such mechanisms as Fast Recovery in TCP New Reno to detect multiple packet losses within a single window.

VII. Performance Comparison

We evaluated the performance of ITP, TCP (New Reno), and NDN using the ns3 [66] and ndnSIM [44] simulators, and considered the efficacy of congestion control methods and the efficiency of transparent caching.

A. *Efficacy of Congestion Control*

We compare the congestion control algorithm and retransmission policies of ITP, TCP and NDN using a scenario consisting of a simple network consisting of a single source and a single sink. We assume that NDN uses an end-to-end protocol that behaves in the same way as TCP to provide a fair comparison. Accordingly, consumers in NDN can only infer congestion via a retransmission timeout and use AIMD window control to avoid congestion, such a mechanism is used by most end-to-end protocols in ICN. The topology of the network is a single path of four nodes with a single sink at one end and a server at the other end. Both ends share a common bottleneck of 1.5 Mbps. For a fair comparison, no in-network caching for ITP takes place in this scenario. The size of the object chunks in ITP and NDN are equal to the segment size in TCP, and fixed at 1500 Byte. Both ITP and TCP share the same fixed header size, and we use a short content name in NDN to avoid additional overhead in NDN due to large names.

Table I shows the results for average throughput, packet loss, total download time, and jitter for a file of 3.7MB being downloaded. The overall completion time for NDN is worse compared to TCP and ITP because a consumer in NDN cannot detect the data source, which prevents the use of out-of-order delivery methods to detect packet losses. Accordingly, consumers must rely on methods that depend on retransmission timeouts [7], [13].

ITP detects and recovers from a packet loss faster than TCP because of its retransmission policy. As Table I shows, the completion time for TCP is higher than in ITP. This is mainly due to the fact that ITP does not use connections and applies a fast retransmission strategy enabled by Manifests. It takes TCP a minimum of 1 RTT to start sending data while it takes, ITP only half the RTT. In addition, when TCP closes a connection, both ends must terminate the connection even if only one of the ends was transmitting data. However, for ITP, only the consumer requires to signal the producer that the complete data was received. In addition, multiple packets lost within a single-window can affect TCP performance even with the SACK option enabled. Because ITP is receiver driven, the consumer has a complete picture of which data packets were received correctly and which one was lost, and does not rely on partial ACK's like TCP does. Accordingly, it immediately goes into congestion avoidance state, instead of fast recovery. As a result, ITP continues increasing its congestion window normally. This gives ITP the advantage of utilizing the bottleneck's buffer compared to TCP, especially under shallow-buffer scenarios [6].

TABLE I
SINGLE-FLOW RESULTS

	ITP	TCP	NDN
Total time (sec)	33.2694	35.5447	35.9408
Avg. throughput (Mbps)	1.41298	1.34652	1.2736
Packet loss	39	59	56
Jitter sum (sec)	3.3151	4.2827	4.9951

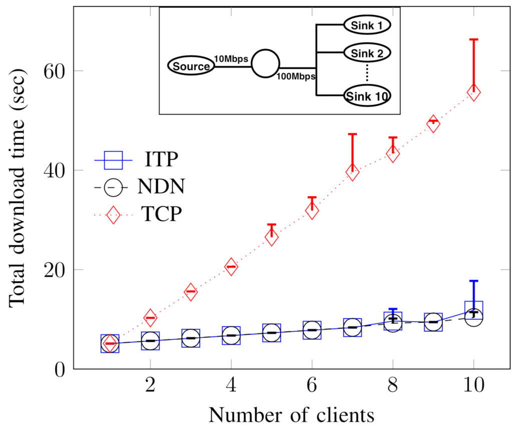

Fig. 4. Total transfer time vs. number of sinks

B. *Efficiency of Transparent Caching*

We compared the total time taken to retrieve multiple copies of a large data file using ITP, TCP, and NDN. This scenario highlights the ability of ITP to take advantage of transparent caching without requiring any changes to the communication infrastructure. The experiment assumes a network consisting of a source node connected over a 10 Mbps shared link to a cluster of 10 sink nodes, all interconnected via 100 Mbps links, where a middle node in this scenario is acting as a caching proxy for ITP traffic. The same topology was used for NDN as well. For a fair comparison between NDN and the other protocols, we used the same transport protocol highlighted in the previous scenario. We ran ten scenarios; with each scenario, we increased the number of sinks in the network, bringing the total to 10 sinks. Each sink starts pulling a 6MB data file from the source at random start time based on a Poisson distribution with an average arrival of 5 minutes. We ran each scenario ten times, each one with a different random arrival time. The total elapsed time for all the sinks to complete the task was recorded and displayed in Fig. 6.

TCP, ITP, and NDN perform much the same when only one sink is involved, given that most requests in ITP and NDN are retrieved from the source, and the three approaches use very similar algorithms for congestion control. As the number of sinks increases, the completion time in ITP and NDN remain fairly constant while the completion time in TCP increases linearly because all data have to be retrieved from the source. NDN outperforms ITP when two or more downloads start before data are available at the nearby cache. In ITP this results in those Interests being sent to the producer while in NDN only the first Interest is sent.

VIII. CONCLUSIONS AND FUTURE WORK

We introduced ITP, the first connection-free reliable transport protocol. Its design consists of the integration of a message-switching approach first discussed by Walden [67], [68] with the use of manifests [40] and receiver-driven requests [30]. ITP eliminates the need for servers to maintain per-client state and allows for all application data to be cached transparently on the way to consumers using ITP Caching Proxies. To prevent such middle boxes from accessing cached content, ITP can rely on encoding of data objects in ways that can only be understood by those consumers who have the manifests needed to decode the data objects.

Much has been written about the inability of the current IP Internet architecture to support the emerging Internet content-oriented applications. Our description and analysis of ITP demonstrates that the IP Internet can provide efficient support of such applications by means of connection-free reliable transport services. We hope that our work inspires the reader to propose new ways to provide a content-centric solution based on an IP Internet that is free of connections.

REFERENCES

[1] Y. Afek et al., "Reliable Communication over Unreliable Channels," *JACM*, Nov. 1994.

[2] A. Agrawal, "Xavier: A Reinforcement-Learning Approach to TCP Congestion Control," Technical Report, Stanford University, 2016.

[3] B. Ahlgren et al, "A Survey of Information-Centric Networking," *IEEE Commun. Magazine*, July 2012, pp. 26–36.

[4] A. A. Albalawi et al, "A Delay-Based Congestion-Control Protocol for Information-Centric Networks," *IEEE ICNC '19*, 2019.

[5] A. A. Albalawi, and J.J. Garcia-Luna-Aceves, "Named-Data Transport: An End-to-End Approach for an Information-Centric IP Internet," *Proc. ACM ICN '20*, 2020.

[6] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," *Proc. ACM SIGCOMM '04*, Aug. 2004.

[7] S. Arianfar, et al., "Contug: A Receiver-Driven Transport Protocol for Content-Centric Networks," *Proc. IEEE ICNP '10* (Poster session), 2010.

[8] A.E. Baratz and A. Segall, "Reliable Link Initialization Procedures," *IEEE Trans. Commun.*, February 1988.

[9] M.F. Bari et al, "A Survey of Naming and Routing in Information-Centric Networks," *IEEE Commun. Magazine*, July 2012, pp. 44–53.

[10] D. Belsnes "Single-Message Communication," *IEEE Trans. Commun.*, Feb. 1976.

[11] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, 1992.

[12] S. Braun et al., "An Empirical Study of Receiver-based AIMD Flow Control for CCN," *Proc. IEEE ICCCN '13*, 2013.

[13] G. Carofiglio, M. Gallo, and L. Muscariello, "ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking," *Proc. IEEE NOMEN '12*, March 2012.

[14] G. Carofiglio et al., "Multipath Congestion Control in Content-Centric Networks," *Proc. IEEE NOMEN '13*, April 2013.

[15] S. Carr, S.D. Crocker, and V.G. Cerf, "HOST-HOST Communication Protocol in The ARPA Network," *Proc. Spring Joint Computer Conference '70*, 1970.

[16] Q. Chen et al., "Transport Control Strategies in Named Data Networking: A Survey," *IEEE Communications Surveys and Tutorials*, 2016.

[17] S. Cheshire, J. Graessley, and R. McGuire, "Encapsulation of TCP and other transport protocols over UDP," Internet Draft, Jul. 2013.

[18] V.G. Cerf and R.E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Trans. Commun.*, May 1974.

[19] V.G. Cerf, Y.K. Dalal, and C.A. Sunshine, "Specification of Internet Transmission Control Program," INWG Note 72, revised Dec.1974.

[20] V.G. Cerf, "Specification of Internet Transmission Control Program, TCP (Version 2)" March 1977.

[21] V.G. Cerf and J. Postel, "Specification of Internet Transmission Control Program, TCP (Version 3), Jan. 1978.

[22] A. Dabirmoghaddam et al., "Characterizing Interest Aggregation in Content-Centric Networks," *Proc. IFIP Networking '16*, May 2016.

[23] A. Dabirmoghaddam et al., "Understanding Optimal Caching and Opportunistic Caching at the Edge of Information-Centric Networks," *Proc. ACM ICN '14*, Sept. 2014.

[24] D.W. Davis, D.L.A. Barber, W.L. Price, and C.M. Solomonides, *Computer Networks and Their Protocols*, Wiley & Sons, 1979.

[25] T. Dierks. The transport layer security (TLS) protocol version 1.2. 2008.

[26] B. Ford, "Structured Streams: A New Transport Abstraction, *Proc. ACM SIGCOMM '07*, Aug. 2007.

[27] A. Ford et al., "Architectural guidelines for multipath TCP development," IETF, RFC 6182, Mar. 2011.

[28] Y. Gu and R. Grossman, " UDT: UDP-Based Data Transfer for High-Speed Wide Area Networks," *Computer Networks*, Elsevier, Volume 51, Issue 7, 2007.

[29] V. Jacobson, "Congestion Avoidance and Control, *Proc. ACM SIGCOMM '88*, Aug. 1988.

[30] V. Jacobson et al., "Networking Named Content," *Proc. ACM CoNEXT '09*, 2009.

[31] V. Jacobson et al., "VoCCN: Voice-over Content-Centric Networks," *Proc. reArch '09*, Dec. 2009.

[32] N. Jay et al., "A Deep Reinforcement Learning Perspective on Internet Congestion Control," *Proc. 36th Int' Conf. on Machine Learning*, 2019.

[33] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *Computer Communication Review*, volume 17 No. 5, August 1987.

[34] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, IETF, March 2006.

[35] A. Langley et al., "The QUIC Transport Protocol: Design and Internet-Scale Deployment," *Proc. ACM SIGCOMM '17*, Aug. 2017.

[36] S. Loreto et al., "Explicit Trusted Proxy in HTTP/2.0," IETF Internet-Draft, 2014.

[37] N. A. Lynch, Y. Mansour, and A. Fekete, "Data link layer: two impossibility Results," *Proc. ACM PODC '88*, 1988.

[38] N.A. Lynch, *Distributed Algorithms*, Morgan Kauffman, 1996.

[39] I. Moiseenko "Fetching content in Named Data Networking with embedded Manifests" 2014.

[40] M. Mosko, "CCNx Manifest Specification," 2016

[41] D. Naylor et al., "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS," *Proc. ACM SIGCOMM '15*, Aug. 2015.

[42] M. Nowlan et al, "Fitting Square Pegs Through Round Pipes: Unordered Delivery Wire-Compatible with TCP and TLS," *USENIX NSDI '12*, 2012.

[43] NSF Named Data Networking project. [Online]. Available: http://www.named-data.net/

[44] NS-3 based Named Data Networking (NDN) simulator [Online]. Available: https://ndnsim.net/current/index.html

[45] G. Papastergiou et al., "De-ossifying the Internet Transport Layer: A Survey and Future Perspectives," *IEEE Communications Surveys and Tutorials*, Nov. 2016.

[46] C. Parsa and J.J. Garcia-Luna-Aceves, "Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media," *Proc. IEEE ICNP '99*, Nov. 1999.

[47] R. Peon, "Explicit Proxies for HTTP/2.0," IETF Informational Internet Draft, 2012.

[48] D. Perino and M. Varvello, "A Reality Check for Content Centric Networking," *Proc. ACM ICN '11*, 2011.

[49] M. Polese et al., "A Survey on Recent Advances in Transport Layer Protocols," *IEEE Communications Surveys and Tutorials*, Aug. 2019.

[50] J. Postel, "DoD Standard Internet Protocol," IEN 128, RFC 760, Jan. 1980.

[51] J. Postel, "DoD Standard Transmission Control Protocol," IEN 129, RFC 671, Jan. 1980.

[52] J. Postel, C.A. Sunshine, and D. Cohen, "The ARPA Internet Protocol," *Computer Networks*, 1981.

[53] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. IETF RFC 3168, September 2001.

[54] R. Rivest, "All-or-nothing Encryption and the Package Transform," *Proc. Fast Software Encryption*, 1997.

[55] J. Rosenberg, "UDP and TCP as the New Waist of the Internet Hourglass," IETF Internet Draft, 2008.

[56] N. Rozhnova et al., "An Improved Hop-by-hop Interest Shaper for Congestion Control in Named Data Networking," *Proc. ACM ICN '13*, 2013.

[57] L. Saino, C. Cocora, and G. Pavlou, "CCTCP: A scalable receiver-driven congestion control protocol for content centric networking," *IEEE ICC '13*, 2013.

[58] K. Schneider et al., "A Practical Congestion Control Scheme for Named Data Networking," *Proc. ACM ICN '16* , 2016.

[59] J. Sherry et al., "BlindBox: Deep Packet Inspectionover Encrypted Traffic," *Proc. ACM SIGCOMM '15*, Aug. 2015.

[60] J.M. Spinelli, "Reliable Communication on Data Links," LIDS-P-1844, MIT, Dec. 1988.

[61] R. Stewart, "RFC 4960: Stream Control Transmission Protocol (SCTP)," IETF, 2007.

[62] A. Stubblefield and D. Wallach, "Dagster: Censorship-Resistant Publishing Without Replication," Rice University, Tech. Rep. TR01-380, 2001.

[63] C.A. Sunshine, "Interprocess Communication Protocols for Computer Networks," Ph.D. Thesis, Tech. Report 105, Stanford University, Dec. 1975

[64] C.A. Sunshine and Y.K. Dalal, "Connection Management in Transport Protocols," *Computer Networks*, 1978.

[65] B. Trammell, M. Kuehlewind, E. Gubser, and J. Hildebrand, "A New Transport Encapsulation for Middlebox Cooperation," *Proc. IEEE CSCN '15* 2015.

[66] ns-3 Network Simulator [Online]. Available: https://www.nsnam.org

[67] D.C. Walden, "A System for Interprocess Communication in a Resource Sharing Computer Network," *CACM*, April 1972.

[68] D.C. Walden, "Host-To-Host Protocols," in *Tutorial: A Practical View of Computer Communications Protocols* (J.M McQuillan and V.G. Cerf, Ed.s), pp. 172-204, IEEE, 1978.

[69] D. Wand and L.D. Zuck, "Tight Bounds for the Sequence Transmission Problem," *Proc. ACM PODC '89*, Aug. 1989.

[70] K. Winstein and H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control," *Proc. ACM SIGCOMM '13*, 2013.

[71] G. Xylomenos et al., "A Survey of Information-centric Networking Research," *IEEE Communication Surveys and Tutorials*, July 2013.

[72] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A Case for Stateful Forwarding Plane. Tech. Report NDN-0002, July 2012.

[73] H. Zimmermann, "The CYCLADES End-to-End Protocol," *Proc. ACM Fourth Data Commun. Symposium*, Oct. 1975.