A Connection-Free Reliable Transport Protocol

J.J. Garcia-Luna-Aceves
Computer Science and Engineering Department
University of California
Santa Cruz, CA, USA
jj@soe.ucsc.edu

Abdulazaz Ali Albalawi
Computer Science and Engineering Department
University of California
Santa Cruz, CA, USA
aalbalaw@ucsc.edu

2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC) | 978-1-7281-9829-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/IPCCC50635.2020.9391540

***Abstract*—The Internet Transport Protocol (ITP) is introduced as an alternative to the Transmission Control Protocol (TCP) for reliable end-to-end transport services in the IP Internet. The design of ITP is based on Walden's early work on host-host protocols, and the use of receiver-driven Interests and manifests advocated in several information-centric networking architectures. The performance of ITP is compared against the performance of TCP using off-the-shelf implementations in the ns3 simulator. The results show that ITP is inherently better than TCP and that end-to-end connections are not needed to provide efficient and reliable data exchange in the IP Internet.**

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

I. Introduction

The two main transport protocols used in the Internet today are the User Datagram Protocol (UDP), which provides best-effort delivery between remote two processes, and the Transmission Control Protocol (TCP), which provides reliable in-order delivery of data between two remote processes.

The current design of TCP is based on the original proposal by Cerf and Kahn on the *Transmission Control Program* [3], which combined datagram delivery and end-to-end transport functionality in a single protocol. Like most of the early work on reliable transport protocols, the Transmission Control Program used connections to implement reliable in-order delivery of data between remote processes. Since then, even after the original design by Cerf and Kahn [3] was divided into the Internet Protocol (IP) [10] and TCP [11], connections have been used to support end-to-end reliable communication.

Section II provides a critique of prior work related to reliable communication between remote processes over computer networks. What is most notable is that, with one exception [1], no prior transport protocol has been proposed that provides reliable communication without end-to-end connections over a datagram-based communication infrastructure.

This paper introduces the **Internet Transport Protocol** (ITP), which is the first connection-free reliable transport protocol that operates over a datagram-based communication infrastructure and uses existing name-resolution services available in the Internet without modifications.

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.

ITP is based on Walden's message-switching host-host protocol [13], the use of manifests and receiver-initiated requests for content advocated in a number of information-centric networking (ICN) architectures, and the inclusion of pointers to manifests in each request for data and response.

Sections III describes how ITP operates. In a nutshell, an association between two remote processes is established with a two-way handshake in which a process A requests some data from a remote process B by sending a data packet, and process B returns a *manifest* informing process A what Interests (requests) to send to obtain the requested data. Once process A has a manifest, it sends Interests to process B making reference to the manifest and informing B of what A needs. Process B simply responds to Interests from process A, without maintaining any state regarding process A. There is no need for signaling to start or terminate the association; process A can end the association when it obtains all it needs according to the manifest, and process B can simply end the association after a timeout.

Sections IV and V summarize retransmission and congestion control mechanisms for ITP that were chosen to be very similar to those currently used in TCP (TCP Reno) to highlight that ITP is inherently more efficient than TCP.

Section VI compares the performance of ITP and TCP using the ns3 [12] simulator. The results indicate that ITP is inherently more efficient than TCP.

Section VII provides our conclusions and discusses promising future research.

II. Related Work

Walden [13] proposed a message-switching host-host protocol for the ARPANET; however, connections have become the way to support reliable communication between remote processes in the Internet.

Considerable work has been reported over the years on transport protocols that provide reliable end-to-end communication [8], [9], and most reliable transport protocols use byte streams as the abstraction for retransmission and congestion-control.

A plethora of proposals have been made to improve the performance and functionality of TCP, including more efficient flow control and retransmission strategies, the use of multiple

connections, enabling multihoming of the remote processes exchanging data reliably (e.g., [7]), and even the use of machine learning in TCP congestion control (e.g., [6]).

What is so striking about the prior work on reliable transport protocols operating over a datagram infrastructure is that, with one exception [1], all of the transport protocols require the use of end-to-end connections to specific addresses. This has become a problem because of the growing need to support location-independent Internet services and content.

Several information-centric networking (ICN) architectures have been proposed to date, and it is worth noting that the congestion-control schemes proposed for such architectures are very similar to those used in TCP [2].

III. Internet Transport Protocol

A. Nexus: Implementing Associations without Connections

For two remote processes to communicate with one another reliably, one must be able to address the other *and* they must understand what they communicate to each other. We call these two requirements of an *association* between two processes as addressing and context. The current use of pairs of IP addresses and ports satisfies the addressing requirement of an association in the IP Internet. But what about the context?

If by necessity the communicating processes have no means of discerning the structure of the data they are exchanging, then they must agree on a method whereby they agree synchronously on the state of their exchange using a common representation for that state. In this light, the evolution of TCP into a connection-based transport protocol makes sense given the state of computing technology back in the 1970's and 1980's when TCP was developed. Implementing an association using a *connection* uses limited computing resources by establishing an agreement for the reliable exchange of a byte stream with an initial sequence number and provides enough context for processes to agree on how many bytes have been successfully delivered. However, the price paid for such efficiency in resource utilization is big in terms of functionality. The context of the reliable communication between the remote processes must be established in real time, because there is no information regarding the structure of the data to be exchanged, and hence must also be updated and terminated jointly. Furthermore, a connection is ephemeral and the context of the association is lost if the connection is broken.

ITP overcomes the inherent functionality limitations of end-to-end connections by establishing *asynchronous* associations between processes that are not lost if physical connectivity is lost. This is done based on two insights: (a) no correct connection-based protocol can be defined if delays are unbounded and nodes may lose connection state; and (b) the names of data objects can be defined to be permanent and uniquely associated with the objects they denote.

ITP takes advantage of the uniqueness and immutability of data objects' names to establish the needed context of an association independently of network delays or the occurrence of node or link failures, out-of-order datagram delivery, or packet replication. 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. Provided that communicating processes can refer to the same manifest, they can exchange elements of the data object described in the manifest on a transactional basis, and a consumer process is free to contact multiple processes using the same nexus provided by the manifest.

We use the term **nexus** to denote the establishment of the context of an association between processes by means of manifests and references to them.

Figure 1 illustrates how a producer transmits a data object D to a consumer in ITP based on a nexus. As in other modern transport protocols, 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 $M(D)$.

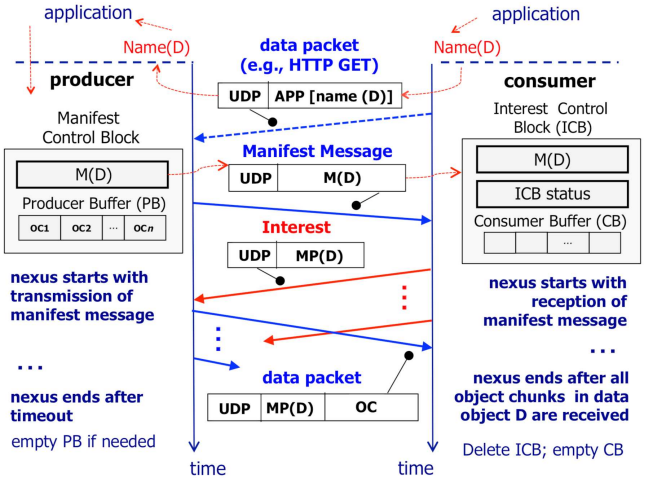


Fig. 1. Nexus in ITP

The application running at the consumer site asks ITP to send a data packet, which requests a data object (e.g., an HTTP GET). In response to that request, the producer specifies a number of parameters through a specific system call, includng the content to send, the IP and port address of the client, and additional control information depending on the system call it used. Rather than setting up a connection to send the data, the ITP process constructs a manifest and sends it to the client as a data packet called *Manifest Message* in Figure 1, and creates a *Manifest Control Block* (MCB) that specifies the manifest $M(D)$, 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.

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 when it receives the manifest $M(D)$. After receiving $M(D)$, a consumer creates an *Interest Control Block* (ICB) for D and allocates memory for a *Consumer Buffer* (CB) to store OC's of D. The ICB includes $M(D)$ and the ICB status indicating the OC's that have been received and those that are missing. The nexus at the consumer ends when it has all the OC's needed for D, at which point it deletes the ICB for D. Once the consumer has a nexus for D, it obtains the data in D by sending *Interests* for D. An Interest for D is denoted by $I(D)$ and states: (a) the names of the consumer and producer; (b) the name of the data object; and (c) a manifest pointer ($MP(D)$) that references $M(D)$ and states implicitly or explicitly the OC's 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.

B. Software Architecture

As Figure 2 shows, the ITP software architecture consists of five data structures: Producer, Consumer, Manifest control block (MCB) list, Interest control block (ICB) list, and Content Store. All entities in ITP have the same structure, making it a bi-directional messaging protocol.

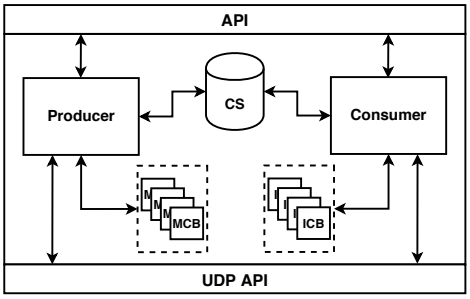


Fig. 2. ITP software architecture

ITP messages are encapsulated in UDP packets, which ensures that no alterations are needed for the network stack and minimal changes are required for the application because most network devices support UDP packets. As a result, ITP can operate completely in user-space, allowing us to rapidly develop and experiment with the protocol without any change to the transport layer. This implementation makes ITP interface on one side to user or application processes and the other side to the UDP interface. The UDP socket is created once the application process creates an ITP socket. Eventually, the UDP module calls on the Internet module to transmit each segment passed by the ITP layer.

Producer & 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 occurred when the client application sent 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 that encapsulates the HTTP GET request instead of constructing a manifest and sending it to the ITP consumer at the server end.

Control Blocks and Content Store: An ITP producer uses the *Manifest Control Block* or MCB to remember several variables for each manifest it creates. Some of these variables represent the manifest itself, the manifest timeout, and consumers authorized to retrieve the data object. These structures are stored in a list. The MCB 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 stores the variables needed for each data object it needs to retrieve in a data structure known as *Interest Control Block* or ICB. As with the MCB, all these structures are stored in a list. 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.

The Content Store (CS) in ITP resembles the NDN content store and TCP send buffers. An ITP producer can use object chunks (OC) in the 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 CS stores the OC's based on their names.

C. Data Naming and Transparent Caching

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 incriminating it until it receives all the data packets with the OC's of the data object. An alternative way to name OC's in ITP is by using a cryptographically secure hash function of their content. The hash digest for OC's in a manifest can be thought of as the sequence number in TCP used to identify streams of bytes. This means that a manifest is a data object with a hash digest as its name that describes an ordered collection of OC's and their corresponding hash digest. The combination of the OC's stated in the manifest and carried out according to the method indicated in the manifest renders the original data object.

Large data objects can be organized into a hierarchy of decoding manifests. To prevent fragmentation and reassembly, an OC should be small enough to fit in any link-level frame.

ITP can support transparent caching by having a data object name prefixed with the ITP producer's IP address and application's port number. This makes OC names to be globally unique and prevents hash collision at in-network caches.

D. Application Dialogue over ITP

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. Given that no connection is established from the client to the server, the client just sends messages to the server using a `FORCE_SEND()` call that 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 a system call `SEND()`. It is up to the application dialogue 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.

IV. Retransmission Strategy

ITP uses a receiver-driven selective-repeat retransmission strategy inspired by the work by Jacobson et al. [5] and based on the fact that both consumer and producer have the manifest of the data object being exchanged.

Consumer Steps: The consumer keeps track of the status of its control buffer (CB) to ensure that a data object is passed to the application correctly. The initial status of the CB is set with all OC's missing when the consumer sends its first Interest to the producer for a data object D, and the CB status is updated with each OC received correctly from the producer. Depending on the application using ITP, the consumer may have to receive all the OC's of a data object before it passes them to the application process and then empty its ICB. Alternatively, the consumer may be allowed to pass to the application OC's that are in order and without any missing OC's, and delete those OC's from its CB.

The consumer is in charge of retransmissions, and simply persists sending Interests to the producer asking for the OC's needed to decode the data object, and each Interest includes a manifest pointer that reflects the latest snapshot of the CB status. As such, each Interest can be viewed as including a vector of acknowledgments informing the producer about the OC's that have been received correctly and those that are missing. The consumer maintains a list of Interests based on the local times when they were transmitted. The manifest pointer of each Interest includes the local transmission time or a sequence number that differentiates it from any other Interest, even when multiple Interests carry manifest pointers stating the same CB status. Hence, each data packet received can be matched uniquely with a transmitted Interest and the consumer can accurately update its round-trip time (RTT) estimate with every data packet it receives, even when data packets and Interests are lost or delivered out of order.

To facilitate efficient retransmissions, the consumer applies a retransmission timeout (RTO) for each Interest it sends to the producer; the RTO is updated based on Jacobson's algorithm [4]. This allows ITP to obtain better RTO estimates by measuring the correct RTT with each data packet. The consumer retransmits an Interest i originally sent at time t_i in two cases: (a) after the RTO for Interest i expires, or (b) when a data packet for an Interest j sent at time $t_j > t_i$ arrives at time t_d and $t_d - t_i >$ RTT. The latter allows for fast retransmits without incurring unnecessary Interest retransmissions.

Producer Steps: A producer simply responds to Interests from any consumer regarding a data object D using the information about D stored in its MCB and the manifest pointer included in each Interest. The manifest pointer carried in an Interest identifies the consumer and the OC's that it needs. The order in which a producer receives Interests is not critical, and the occurrence of Interest losses or Interest duplicates does not confuse a producer, because each Interest carries a manifest pointer.

V. Congestion Control

We describe a congestion-control strategy for ITP that mimics the approach used in TCP Reno, but takes advantage of the receiver-driven retransmission strategy using manifest pointers in Interests and data packets. Using a congestion-control strategy that mimics TCP Reno allows us to illustrate the inherent benefits of the connection-free receiver-initiated approach used in ITP compared to the connection-based sender-initiated approach used in TCP.

We adopt a simple Interest-based approach for congestion control in which one Interest from the consumer elicits one data packet from the producer. This is the original approach advocated in CCN [5] and NDN [**?**]. However, more sophisticated approaches could and should be implemented in which

a window of packets or packets with multiple OC's are sent in response to Interests.

Producer Steps: Data packets are the main cause of congestion in ITP, because the size of an Interest is relatively small compared to a data packet. The use of receiver-driven Interests allows the data traffic in ITP to be controlled by controlling the rate at which the consumer issues Interests. This frees the producer from having to maintain any per-consumer congestion state. Accordingly, the producer's role in congestion control is minimum and consists of simply submitting requested OC's upon reception of Interests from consumers.

Consumer Steps: The consumer controls the flow of data traffic with the producer by controlling its Interests' sending rate using a congestion window. The congestion window defines the maximum number of outstanding Interests allowed to send without receiving their data packets.

The window size is adjusted by the consumer based on the AIMD (Additive Increase Multiplicative Decrease) mechanism commonly used in TCP for the congestion window. Similar to TCP Reno, the consumer in ITP starts in slow-start with a congestion window of size one. The value of the congestion window size is increased for each data packet received. This continues until either a loss is detected or the congestion window reaches the slow-start threshold, ssthresh. Once the consumer exceeds the ssthresh, it enters the congestion avoidance state as in TCP [4]. During this state, the consumer increases its congestion window by one Interest every round-trip time.

When an Interest times out, the consumer retransmits the Interest and reduces its congestion window to one. It sets the ssthresh to half the congestion window size before the timeout, and then goes into slow-start. If it detects a packet loss using fast retransmit, the consumer reduces its congestion window by one half and sets the ssthresh to the new window size causing it to go into congestion avoidance.

VI. Performance Comparison

We evaluated the performance of ITP, TCP (New Reno), and NDN using the ns3 [12] asimulator, and considered the efficacy of congestion control methods and fairness.

A. Efficacy of Congestion Control

We compared the congestion-control and retransmission mechanisms of ITP and TCP assuming a scenario consisting of a simple network of a single source and a single sink.

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 and no in-network caching takes place. The propagation delay between the two ends is set to $40ms$. Consumers in ITP issue Interests for the content served at the other end, where the client in TCP consume traffic generated by the server. The size of the object chunks in ITP are equal to the segment size in TCP, and fixed at 1500 bytes. Both ITP and TCP share the same fixed-header size.

Figure 3(a) shows the evolution of the congestion window every $50ms$ for the three protocols during the first $30s$ of the downloading a content of $3.69MB$, a total of 2465 chunks/segments. The growth of the congestion windows for all three protocols matches the expected behavior of the AIMD algorithm. However, a consumer in NDN cannot detect the data source, which prevents the use of out-of-order delivery methods to detect packet losses.

The ITP retransmission policy allows receivers to detect and recover from a packet loss faster than in TCP, where it took the consumer a total of 35.5s to download the file. This is mainly due to the fact that ITP does not use connections and applies a fast retransmission strategy enabled by manifests.

A consumer in ITP has a complete picture of which OC's were received correctly and which were 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, where it has to go into fast recovery, during which the sender can only transmit new data for every duplicate ACK received. Figure 3(c) shows the queue size of the bottleneck's buffer for only five seconds of the simulation to highlight the idle periods of each protocol. It can be seen from the figure that TCP has more extended idle periods compared to ITP. As a result, ITP achieved higher average throughput due to better utilization of the link's capacity and the buffer size.

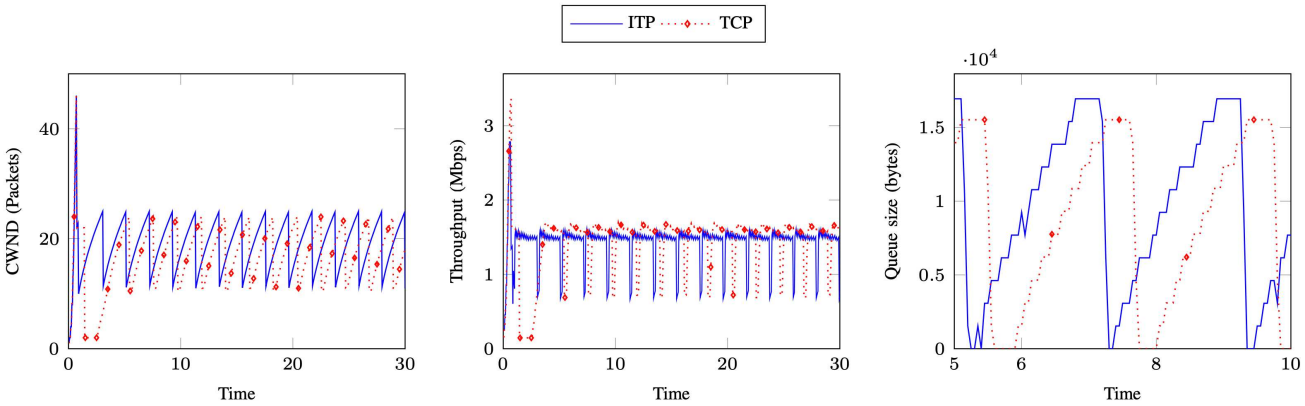
Table 1 shows the average simulation results for TCP and ITP for the same scenario with few changes to the topology. The round-trip time between the two ends is set to $60ms$. The bottleneck of the topology is set to 10Mbps. The access links at the server, and the client is set to 100 Mbps. The client requests/receives data for the duration of the simulation, which is $30s$. The start times of the client's session are randomly chosen between 1s and 5s. The simulation was repeated 10 times, and results were measured at the client-side. The simulation time does not take into account the TCP handshake used to close a connection. As shown from the table, the consumer in ITP achieves higher throughput than TCP, resulting in more bytes received. This is mainly due to ITP connectionless nature, which gives it the ability to detect and react to congestion faster than TCP, resulting in fewer packet losses.

Protocol	Throughput (Mbps)	Bytes Received (Mbyte)	Packet Loss
ITP	9.020282 $\pm$0.01438	31.85985 $\pm$1.84591	115.5 $\pm$0.52705
TCP	8.110141 $\pm$0.05913	28.83405 $\pm$1.75277	220 $\pm$0.00000

TABLE I
SINGLE-FLOW RESULTS

B. Fairness

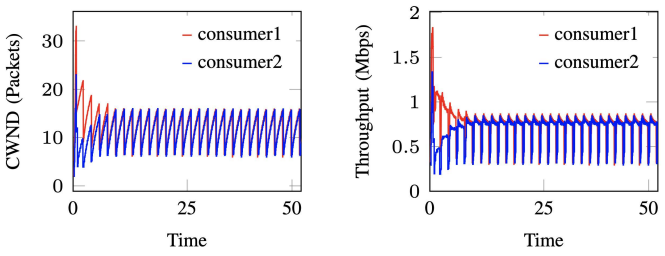
We evaluated fairness among flows in ITP assuming a topology consisting of two consumers and two producers



(a) CWND Size (b) Throughput (c) Router's Buffer Occupancy

Fig. 3. Single-flow scenario for TCP and ITP

connected via a bottleneck link with 1Mbps capacity. The queue size is set to 20 packets and the file size is 10 MB. Both producers transmit the manifests for the file at the beginning of the simulation at the same time. Both consumers start issuing Interests to retrieve the data from the producer after receiving the manifests. We used Jain's fairness index F as our performance metric for this scenario which is defined as $F = (\sum_{i=1}^{n} x_i)^2 (n \sum_{i=1}^{n} x_i^2)^{-1}$, where x_i is the throughput for the ith connection and n is the number of users sharing the same bottleneck resource. The fairness index F is bounded between $1/N$ and 1, where 1 corresponds to the case in which all N flows have a fair allocation of the bandwidth (best case), and $1/N$ refers to the case in which all the bandwidth is given to only one user (worst case).



(a) CWND Size (b) Throughput

Fig. 4. Multiple-flows with homogeneous start

Figure 4(a) shows the evolution of CWND for both consumers. As seen from the figure, ITP CWND behavior follows the usual TCP sawtooth behavior since both are based on an AIMD congestion-control algorithm. The fairness between the two flows is $F = 0.99$; this can also be seen from Figure 4(b), where both consumers achieve a similar average throughput.

VII. Conclusions and Future Work

We introduced ITP, the first connection-free reliable transport protocol that operates using the existing datagram communication infrastructure and name resolution services of the IP Internet. ITP integrates the message-switching approach first discussed by Walden [13] with the use of manifests and receiver-driven requests for content.

We chose to make the retransmission and congestion-control algorithms used in ITP very similar to those used in TCP Reno today to show the inherent benefits of ITP over TCP; however, much more efficient algorithms developed recently for TCP can be adapted to ITP. Furthermore, the use of manifest pointers in ITP enables far more efficient congestion and retransmission control by providing receivers with more control for how and when transmissions and retransmissions should occur.

ITP allows for all application data to be cached transparently on the way to consumers using ITP caching proxies, and this is an area for further study.

References

[1] 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.

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

[3] V.G. Cerf and R.E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications*, May 1974.

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

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

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

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

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

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

[10] J. Postel, "DoD Standard Internet Protocol," RFC 760, Jan. 1980.

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

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

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