A Demo of Viscous: An End-to-end Transport Protocol for Ubiquitous Communication

Abhijit Mondal, Sandip Chakraborty
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Kharagpur, India 721302
Email: abhimondal@iitkgp.ac.in, sandipc@cse.iitkgp.ernet.in

***Abstract*—The existing transport protocols designed for the TCP/IP protocol stack lack three major requirements for ubiquitous communication – support for short-lived parallel flows, mobility and multi-homing. In this demo, we discuss the fundamental design goals and architecture for a new end-to-end communication protocol, called Viscous, that supports the above three requirements. We use a small scale prototype testbed to demonstrate the working principles, implementation details and performance parameters of Viscous for supporting end-to-end communication over Internet of Things (IoT). The complete protocol details for Viscous has been accepted for publication at IEEE LCN 2017 [1].**

I. Introduction

The Internet, which fuels the boom of smartphones and online services, is designed decades ago with the goal of connectivity between devices. It has two primary components – i) Internet Protocol (IP) that provides communication between remote devices, and ii) Transmission Control Protocol (TCP) that supports reliable data communication between two remote applications. In TCP/IP protocol suite, TCP and IP are tightly coupled and suitable for transferring large data from one static system to other remote static system. However, end devices are now mobile with multiple network interfaces and transmit small chunks of data with short-lived parallel flows [2]. This limits the capability of TCP along with the entire network stack.

As mentioned, TCP has three fundamental limitations. i) *Mobility*: Support for seamless connectivity while a device is in move. TCP is tightly coupled with the underlying IP. TCP cannot continue the connection if the underlying IP address gets changed due to mobility [3]. ii) *Multi-homing*: If a device is connected via multiple network interfaces, it is called multi-homed device. TCP can not utilize multiple interfaces simultaneously to transmit data. iii) *short-lived flows*: When a TCP connection does not live long enough to get out of slow-start phase, the flows are called short-lived flow. As TCP increases speed gradually, it cannot utilize full network capability when the data is transferred over multiple short-lived connections [4]. Most of the traffic is being generated by smartphones, smart-devices, and IoT infrastructures, and such ubiquitous Internet traffic are in general short-lived.

A number of recent research work revisited the TCP design so that it can utilize full network capability for both short-lived and long-lived flow. *Multi-Path TCP (MPTCP)* [5] have been developed to support multihoming and mobility. However, it is found that MPTCP does not perform well with short-lived flows [5]. To improve network performance for short-lived connections especially for browsing, Google developed *SPDY* [6] and *QUIC (Quick UDP Internet Connection)* [7]. SPDY suffers from Head of the Line (HoL) blocking as it multiplexes multiple HTTP streams over a single TCP connection. To overcome this problem, QUIC uses UDP as the underlying protocol. QUIC supports mobility as it does not depends on underlying IP address. However, it depends on system's path selection algorithm. So, it is not truly multi-path protocol.

As a consequence we have developed a transport protocol, called *Viscous*, which supports all the limitations of the existing transport protocols. It is developed at user level on top of UDP as transport layer protocol, similar to Google's QUIC. It can multiplex a number of flows through multiple network interfaces available to a device. We have implemented it as a Linux application library and have tested the performance on an emulated network using `Mininet`. In this demo, we show the design principles of Viscous, along with its architecture and implementation fundamentals. Our demonstration is based on a emulation platform (to test the scalability of the system) as well as over a real IoT testbed developed using Raspberry Pi devices.

II. Viscous: the Protocol Fundamentals

Viscous is designed to overcome the limitations of the existing transport layer protocols. It is a UDP based multi-path multi-flow protocol, which is connection-oriented but independent of the network layer. It works as a middleware or wrapper between the user application and the network protocol stack. The details of Viscous protocol are available in the full paper [1] and in online[1]. The design goals for Viscous are as follows.

1) Viscous mitigates the signaling overhead associated with connection establishment, when the data are being transferred over a large number of short-lived flows. This will provide zero round trip time (RTT) connection establishment, which in turn reduces the packet transmission time.

[1]https://abhimp.github.io/Viscous (last accessed: July 25, 2017)

2) It avoids slow-start phase for all the flows, rather, we can multiplex multiple flows over a single connection by maintaining congestion control independent of flow control. This allows flows to start communicating without any independent and individual slow-start.
3) Viscous is able to utilize multiple network interfaces available to a device.
4) Viscous supports mobility, and it does not suffer from HoL blocking while multiplexing multiple flows.

A. Key concepts

To achieve the design goals, we develop Viscous on top of UDP with two key concepts, i) *Channel* and ii) *Flow*.

1) *Channel*: Channels are the individual connection through a path. There is one single channel for each path. Every channel has its own congestion control algorithm and loss recovery algorithm. Although channel provides reliable communication over two hosts, it does not provide ordered delivery assurance. Unlike TCP, channel works on packet instead of byte-streams. It delivers a packet to the flow handler whenever it receives one.
2) *Flow*: Flows are responsible for ordered data communication between a source and a destination, while maintaining flow control. Flows use channels to transmit data. It takes byte-stream from an application, packetizes them, and forwards those packets to channel scheduler to be delivered via one of the channels. On the receiver side, flows reorder the packets it receives from the channels. This way, Viscous avoids HoL blocking. Channels do not wait for lost packets. If one packet from a flow gets lost, other flows do not have to wait for it.

Channel can be established over each compatible pair of IP in between two hosts. Each flows in the application can utilize all the channels for transmission of data.

With channel and flows, we achieve flow multiplexing over different paths, as these are independent of each other. We use UDP as the underlying transport protocol, because UDP is a stateless, sessionless protocol, which also provides best effort packet transmission between two remote hosts. It allows us to develop our own congestion control and reliability modules. It helps us avoiding HoL blocking. Unlike TCP, a channel does not provide order delivery guaranty. So, if a packet loss occurs in a channel, it does not hold other packets from being delivered in time. So, only one flow suffers due to a packet loss, not all subsequent flows.

Channel and flow schemes also allow us to separate congestion control and flow control. We do this, because flow control depends on data generation. As channels multiplexes multiple flows with different data generation rate, and it delivers all the packets to flows as soon as it receives it. There is no requirement to make a flow control here.

Channels are also helpful in supporting mobility. When devices are on the move, channels can be disconnected and reconnected as per network state without interruption on any flow. Viscous supports different types of mobility, like ii) *Individual Mobility*, i)*Connect-time Mobility* and iii) *Simultaneous Mobility* [3].

B. Modules

For ease of implementation, Viscous is developed using a modular architecture. There are different modules for different components of Viscous. Among them, three modules are crucial. These are – i) flow handler, ii) channel handler and iii) channel scheduler.

1) Flow Handler: Flow handler handles a flow. When an application sends data, it passes that data to flow handler as byte-streams. Flow handler packetizes them and forwards them to the channel scheduler. It does not store any packet, however, it tracks the packets that has been sent to maintain flow control. When a packet is received, it is flow handler's job to reorder it and to send it to the applications.

2) Channel Handler: Channel handler is another important part of Viscous design. It provides guaranteed packet delivery mechanism, and it also handles congestion. For congestion control, we use modified TCP new Reno with SACK option. A channel works on packets instead of bytes, and so, sequence numbers are per packet based. Whenever a receiver sends duplicated acknowledgments (ACKs), it includes the original sequence number for which this duplicate ACK is triggered. When the sender receives a duplicate ACK, it comes to know about the packets which are received by the receiver so far. This saves many unnecessary retransmission.

3) Channel Scheduler: Channel scheduler decides the path to be taken for each packet. It connects channels to flows. We use a scheduling policy based on self-clocked scheduling. Whenever a channel is free to send some packets as per its congestion control algorithm, the scheduler schedules the next packet to this channel. It is efficient because packets are being scheduled as per channels' capabilities.

III. Experimental Evaluation

We have performed several experiments to evaluate the performance of Viscous. We choose different network topologies on `Mininet` network emulator. Experiments are performed with and without the presence of background traffic. Due to space constraint, we are presenting only one scenario and limited results[2]. In the topology (Fig. 1), *H1* and *H2* are the source and destination. There are four paths available between these hosts. There are other hosts in the network. We have compared performance with TCP, MPTCP, and QUIC.

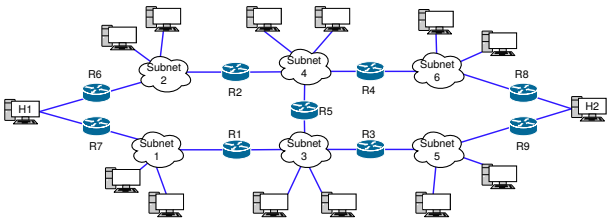
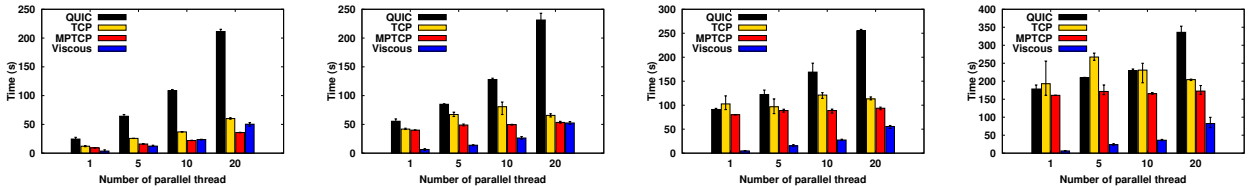


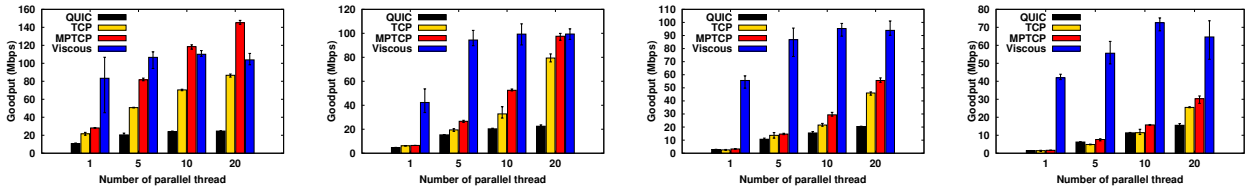
Fig. 1. Topology for Viscous protocol performance evaluation

[2]Detailed results are available at https://github.com/abhimp/Viscous



(a) RTT=16ms (b) RTT=80ms (c) RTT=160ms (d) RTT=320ms

Fig. 2. Average flow completion time for short flows without background traffic



(a) RTT=16ms (b) RTT=80ms (c) RTT=160ms (d) RTT=320ms

Fig. 3. Average goodput for short-lived flows with background traffic

A. Experimental Setup

We have used Oracle VirtualBox to create virtual machines, which have been configured with Mininet kernel to support network interface virtualization. The guest machine configuration is as follows – single core Intel i7-5500U 64bit CPU with clock speed of 2.40 GHz, RAM: 8GB. The host machine configuration is as follows – Intel Genuine i7-5500U 64bit CPU with a clock speed of 2.40GHz, RAM: 8GB with 250GB Solid State Drive. We have used version 2.2.2 of Mininet kernel to develop our test setup. In our virtual machine configurations for test setup, the guest operating system is Ubuntu 14.04.4 LTS, whereas the host operating system is Ubuntu 16.04.1 LTS. The link bandwidth for every path of the topology has been kept as 50 Mbps, and we vary the RTT of the paths from 16ms to 320ms. The background traffic is generated using `iperf` TCP sessions from every host shown in the topology, attached with the various subnets.

B. Performance for short-lived flows

We have compared TCP, MPTCP, and QUIC for a large number of short-lived flows, which is the major drawback for existing protocols. We have sent multiple parallel flows simultaneously using multiple threads. We have used 100 back-to-back flow per thread in the experiment. We have varied data size sent through a flow based on exponential distribution with mean flow size 25KB. The number of parallel flows varied from 1 to 20. the results are shown in Fig. 2 and Fig. 3. From the result, we can see that Viscous performed better when multiple flows are going on simultaneously. QUIC in this scenario does not perform better, because it can not utilize multiple interfaces. Further, it multiplexes the sequential flows from the same thread, but treats parallel flows from different threads as separate flows and maintains separate slow start for them. MPTCP can utilize multiple paths simultaneously. However, it can not perform well because it has to start new connection every time application need to send data through a new flow. TCP also suffers due to the similar reason.

IV. Conclusion

In this demo, we explain the design details for Viscous protocol. Viscous is completely compatible with current network architecture while performing well enough for short-lived connections.

V. Support Required for Demo

The demo will be performed on Mininet environment as well as over a prototype network developed using *Raspberry pi*. For entire setup, we require a standard table and power supply. It will be helpful to display our result if we can get one screen.

References

[1] A. Mondal, S. Bhattacharjee, and S. Chakraborty, "Viscous: An end to end protocol for ubiquitous communication over internet of everything," in *Proceedings of the 42nd IEEE LCN*. IEEE, 2017, pp. 1–9.

[2] J. Rexford and C. Dovrolis, "Future internet architecture: clean-slate versus evolutionary research," *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, 2010.

[3] A. Yadav and A. Venkataramani, "msocket: System support for mobile, multipath, and middlebox-agnostic applications," in *Proceedings of the 24th IEEE ICNP*. IEEE, 2016, pp. 1–10.

[4] G. De Silva, M. C. Chan, and W. T. Ooi, "Throughput estimation for short lived TCP cubic flows," in *Proceedings of the 13th ACM MobiQuitous*. ACM, 2016, pp. 227–236.

[5] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proceedings of the 35th Annual IEEE INFOCOM*. IEEE, 2016, pp. 1–9.

[6] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDYier mobile web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[7] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with QUIC: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.