2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Synchronizing Multiple Data Streams by Time For Vehicle Control

Sid Masih*, Jordan Hart*, Parth Singhal*, and Sascha Hornauer†
*UC-Berkeley, †UC-Berkeley (International Computer Science Institute)
*{sid.masih, jordanhart, parthsinghal}@berkeley.edu †saschaho@icsi.berkeley.edu

***Abstract*—New research on autonomous driving is beginning to incorporate remote, relative-to-vehicle, environmental data. Few protocols exist to deliver remote data to autonomous vehicles — for example observation meta-data or sensor streams such as camera, LIDAR, etc. Researchers bypass this problem by using a single Robot OS controller or rudimentary streaming pipes. However, with recent advancements in vehicle to everything (V2X) protocols, including better LTE and millimeter wave (MMwave) communications in mobile environments, a dedicated multiple-source streaming protocol is increasingly necessary. For this reason, we developed a protocol that allows multiple and distinct transmitters of data to stream sensor information to a single receiver with features including session management, encryption, and jitter regulation. The Synchronized Buffer proposed herein seeks to synchronize data frames by their time stamps, in the spirit of the Real-Time Transport Protocol (RTP) used in VoIP. We validated the proposed time synchronization algorithm by streaming data at different simulated data frames per second. Quality of service (QoS) and potential security flaws are also explored. Such a dedicated streaming protocol addresses current problems by allowing vehicles to integrate previously unavailable remote data sources into critical autonomous decision making.**

I. Introduction

Autonomous driving is a complex task that has primarily focused on cars as isolated units with data gathered locally. Current approaches mostly focus on optimizing driving accuracy with only local data. With the increase of potential data collection from remote sources, a new potential exists for sharing data across various actors (cars, infrastructure, fleet control, etc.). However, using real-time streamed remote data as well as synchronizing this data has its difficulties especially when accounting for network and local latency as well as different data throughput rates. As such, our protocol solves this problem by ensuring single stream delivery and regularizing data by time-stamp after delivery.

II. Related Work

A. Alternatives

Related protocols have existed, but are either just for media content [8], or only solves part of the problem, such as streaming data or synchronizing data streams across a level 2 network[28]. Additionally work has been done in securing the authenticity of data [13], non-real-time synchronization for industrial IoT [17]. Additionally, Robot OS does support data synchronization, but as a work around method and requiring every vehicle with in the network to be under a single master Robot OS node. Furthermore, vehicle controller area networks (CANs) have proposed time synchronization protocols [16], but fail to incorporate remote data or remote jitter.

B. V2X

Vehicle to everything (V2X) which combines vehicle to vehicle (V2V) dedicated short range communications (DSRC) communication using the 802.11p standard and usually a cellular vehicle connection, offers a platform to stream data for moving vehicles. While this paper is not focused directly on instantiating and tearing down vehicle-to-vehicle networks, we assume that basic protocols and infrastructure, such as those seen in Direct V2V Communication with Infrastructure Assistance [3] and the following DSRC and Cellular Survey [4], are accessible and provide support to our protocol.

Furthermore, with the advent of stronger V2X technology, especially of the mission critical spectrum [6], millimeter wave DSRC [3], and optimized existing cellular technology for vehicles (C-V2X) [7], any real-time delivery protocol should be able to operate over cellular and/or 802.11p V2V.

C. Wireless

Much work has been done on the actual radio frequency (RF) physical layer. Millimeter wave, defined as RF frequencies at 30 GHz and above, due to its short range and high throughput nature poses a unique opportunity for streaming and other high bandwidth services. Physical optimization [7] [14] [15] as well as hybrid systems of DSRC and cellular [4] [10] form core advancements. Similarly, optimization using existing LTE and LTE-Advanced architecture form the basis for longer and more reliable communications.

III. Example Use Case

The synchronized buffer allows autonomous vehicles to view streamed data, like camera data, from intersections. This infrastructure enhancement allows vehicles to observe opposing incoming traffic that would not be directly visible to the vehicle in question (line of sight). Additionally, vehicles being able to judge for themselves or receive stop metadata instructions based on cameras directly built into infrastructure allows a much stronger and more direct method of communication over existing visual methods, such as the light system. Multicast protocols for vehicle to everything or just vehicle-to-vehicle communication also make such interchanges bandwidth efficient and practically feasible.

IV. Approach

In order to present the streamed data (at different throughput rates) at each unified outputted frame, each component of the unified frame must be from a similar time-stamp. We define synchronizing data by time as each data input d_i collected at time t_i and time t_j where all $\|t_i - t_j\| < \epsilon$ where ϵ where $i \neq j$ and $i, j \in \mathbb{N}$ is a defined constant. We further define an outputted frame to be a stream of data vectors containing the collective information gathered from multiple sources into a single vector.

Our approach integrates these design requirements to build a streaming protocol that incorporates multiple inputs from sources defined as transmitters to a single receiver. These data streams have data frames that are synced by time-stamps. This design paradigm is in the spirit of Real-Time Transport Protocol (RTP) and related application layer protocols that are popular with other streaming applications such as voice over IP (VoIP) [8].

V. Architecture

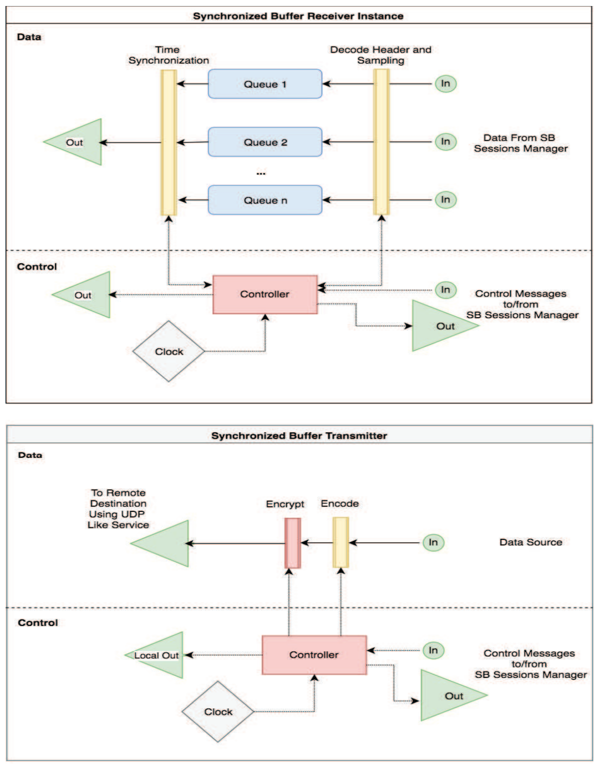
A. Background, Requirements, and Definitions

As an application layer protocol, the synchronized buffer sits above a generalized vehicle to everything (V2X) implementation (the abstractions will be detailed below). As such, drivers would exist for various sub modules for link and network layer communications. The flexibility of generalizing a V2X implementation allows the actors to communicate through different mediums such as ad-hoc V2V VANETS or traditional cellular networks [9]. Therefore, the synchronized buffer (SB) middleware and drivers pass Unix sockets and pipes as abstractions to the underlying networking hardware and firmware. This assumption allows potential adopters to write custom optimized drivers or proprietary software for managing underlying OS and hardware resources for synchronized buffers.

Finally, we assume that there exists a way to prove a vehicles identity on a VANET [11]. Proving identity is essential to preventing Sybil attacks [22] and man-in-the-middle attacks. Since this protocol is not concerned with proving identity, we assume that an IP address or other address correctly corresponds to the intended car or infrastructure endpoint in question.

B. Components

We now define the major system components of the protocol. Transmitter refers to a data source which encodes and transmits streamed data frames to the receiver. We also define a receiver as the destination of all data sent by the transmitters within a session. Our goal is to synchronize data by time as each data input is matched within the epsilon defined in the introduction. Additionally, we define a synchronized buffer de-multiplexer that takes in frames meant for a synchronized buffer instance and routes the reconstructed frame to the correct instance. We define data frames as a single set of data gathered at a distinct time. Finally, we define sockets and pipes according to the Unix POSIX standard [12].



Synchronized Buffer Receiver Instance
Data
Time Synchronization
Decode Header and Sampling
Queue 1
Queue 2
...
Queue n
In
In
In
Out
Data From SB Sessions Manager
Control
Out
Controller
In
Control Messages to/from SB Sessions Manager
Out
Clock
Synchronized Buffer Transmitter
Data
To Remote Destination Using UDP Like Service
Encrypt
Encode
In
Data Source
Control
Local Out
Controller
In
Control Messages to/from SB Sessions Manager
Out
Clock

Fig. 1. Synchronized Buffer Receiver and Transmitter

C. Overall Architecture

We now explore the architecture and intuition for the chosen architecture at a high level. Compared to other protocols, the synchronized buffers overall architecture is designed to minimize dependencies on specific hardware, with abstractions to system calls to traditional cellular/IP based networks as well as vehicle-to-vehicle networks. At the macroscopic level, the synchronized buffer comprises of N queues for N transmitters (both local and/or remote). Each frame is added to the back of a queue given the constraints that will be discussed in Section F. These constraints are designed to enforce a time sorted queue as well as prevent corrupted data from being added to the queue. Figure [2] shows the overarching architecture, while figure [1] shows the transmitter and receiver instances.

We define a header for the data frames, defined as a single related collection of data similar to a packet, to assist with the time matching. Finally the synchronized buffer sessions manager that assists with routing messages and with opening sessions.

D. Data Frame Header Structure

The header, shown in figure [3], is used to store important information about the type of data or encode commands. Note

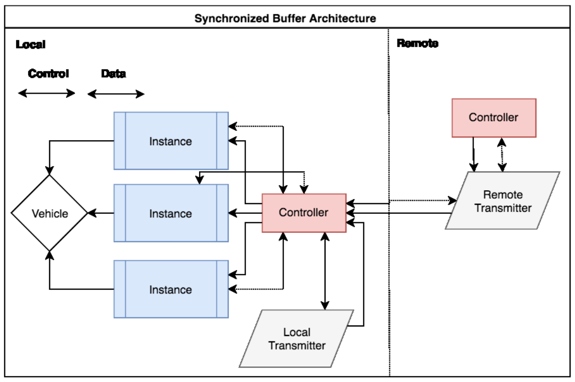


Fig. 2. Macroscopic Diagram

that the header always precedes data and that the size of the data is always defined.

- **Protocol Code**: Decimal 0 for data frame, decimal 1 for control frame, options for other types of synchronized buffers not currently defined but allowed.
- **Session_ID:** Runtime unique serial number identifying a synchronized buffer transmitter and receiver pair. Note that each vehicle may have multiple receiving and transmitting instances.
- **Time-Stamp:** See Section E
- **Flag:** Signals 1 for when time-stamp is rolling over otherwise 0. See Section E
- **Payload Size:** Tells how large the data is in bytes of the data frame arriving. Data frame sizes are fixed per session and set prior to any streaming.
- **Opcode:** Gives an operation code. Only used by control frames. Opcodes are used to open streaming sessions, send acknowledgments (ACKs), as well as other custom messages between two proprietary devices.

Each control frame is sent over a TCP or reliable transport link. Control frames and data are never mixed, meaning data is never sent in a control frame nor are commands given in a data frame. Since control signals are never streamed, the protocol can rely on a slower acknowledgment based on complete and correct delivery. Note that streaming across TCP or other reliable service protocols could introduce significant latency due to ACK based congestion control. Also, redelivering frames which are no longer needed would result in wasted bandwidth and increased latency.

E. Time Matching Algorithm

1) Overview: The time matching algorithm (algorithm [1]) matches the data frames from different sources into a single data frame. From the time matching algorithm each queue has data removed from it and matched with data from a similar time. Note that with pseudo-time, the time-step, again defined as $\|real(t)-real(t')\|$ where t' is an immediate timestep after t and the function $real$ gives the actual time for a pseudo-time argument. Additionally, we define a max difference, ϵ that represents our maximum time-step difference, defined as $argmax(\|time(f_i)-time(f_j)\|), i \neq j$ where f_i and f_j are different queues of data taken at a single pass over the queues, at between the maximum time difference.

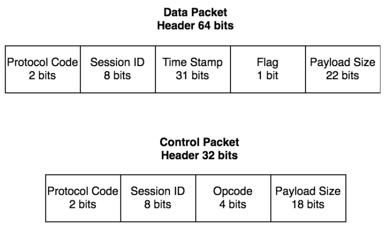


Fig. 3. Data and Control Packet Headers

Before the output frame is generated, a timer starts which will signal a timeout. Next, the algorithm rotates through each queue, collecting parts of the future tuple to be piped from the instance. If a data-part from a queue is older than than the current tuple, the queue is popped until we get data within our ϵ then move towards the next queue. If we find data that has arrived at a time that is too soon, we wait until timeout.

2) Runtime Analysis: The algorithm maximizes performance by minimizing the time spent either revisiting a queue, dropping a frame, or waiting at a queue. As such, each queue is only visited once per tuple generation. This is enforced by only revisiting a queue when a frame is dropped, which causes the next iteration of the algorithm. Therefore, the complexity grows with respect to $Q * |Number\ of\ Frames|$, the number of queues.

F. Pseudo-Time and Time Synchronization

Using RTPs and TCPs idea of counters [8], pseudo-time time-stamps are essential to order streamed data as well as standardize clock differences. The biggest challenge is negating phase-shifted clocks. A phase-shifted clock is defined as clocks C and C' where time $\|C_i - C'_i\| \approx K, K \neq 0$, K being some constant.

One way of synchronizing time is through measuring line latency. During this method, receivers request a pseudo-time from the transmitter. The transmitter then sets its clock to 0 and starts ticking at the desired timestep and sends an ACK with its pseudo-time. The receiver uses this information along with the latency to match time-stamps. If the time-stamp reaches the maximum size in a frames header, we change the time-stamp back to 0 and set a flag in the frame header. Figure [4] details this command flow.

Furthermore, synchronizing using global time through GPS [23] [24] allows the receiver and transmitter to coordinate pseudo-times based on a global time. This system is useful when both devices are equipped with GPS.

The pseudo-time ACK must arrive in less than some user defined sigma (σ) time from the initial request in order for time synchronization to be correctly preformed with σ denoting the

```
Algorithm 1
Given: queues, timer, tick_length, data_out, control_out,
  $\epsilon$, $\gamma$ (list of data frame $\gamma$)
 Assumptions:
  1) frames in queues are in reverse chronological order,
     i.e., from oldest to youngest.
  2) Reference queue, and hence reference frames, is the
     first queue in the list of queues.
reference_queue = queues.pop();
while reference_queue NOT empty do
    tuple = ();
    reference_frame = reference_queue.pop_first_frame();
    for q in queues do
        if not timer.timeout() then
            q_frame = sync_frame(reference_frame, q);
            if q_frame not NULL then
                tuple.append(reference_frame);
                tuple.append(q_frame);
            end
        end
    end
    Output the tuple to data_out;
end
Function sync_frame (reference_frame, queue):
    for frame in queue do
        if
        $\|reference\_frame.time - (frame.time - \gamma_{frame})\| \leq$
        $\epsilon$ / tick_length then
            Remove frame from queue;
            Return frame;
        end
    end
end
```

maximum round trip time (RTT) that is allowed for a packet. The link latency can be derived from dividing the RTT by two. This link latency must be accounted for to avoid large phase shifts in high latency connections. The approximate latency can be measured by measuring the time difference between the initial request to synchronize and the ACK with the transmitters pseudo-time $RTT_{measured}$. The correction value $\gamma = \frac{RTT_{measured}}{2}$ is then added on the receiver to every data frame's time-stamp from the transmitter to correct for connection's latency during matching. Note that V2V networks theoretically have an end to end latency of 1ms, according to 5G and C-V2X requirements [21].

Finally, if using the measured line latency time synchronization method, the protocol assumes that the two transmissions have the same latency, and that the packet is sent back immediately after it is received. It is possible any of these conditions will not hold. If the latency of the transmission from the receiver is larger then the transmission to it, or if the end device waits before sending the packet to the transmitter, the transmitter's correction value will be smaller then the actual difference in clock ticks.

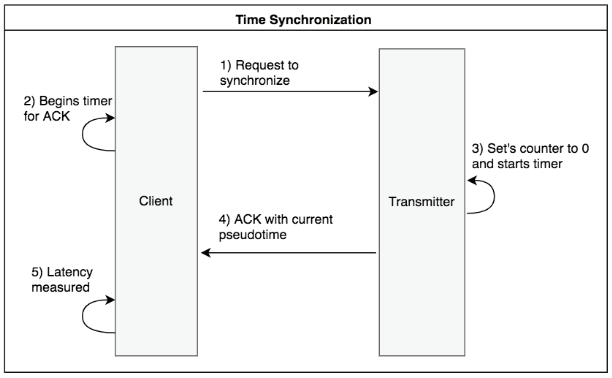

Time Synchronization
Client
Transmitter
1) Request to synchronize
2) Begins timer for ACK
3) Set's counter to 0 and starts timer
4) ACK with current pseudotime
5) Latency measured

Fig. 4. Time Synchronization

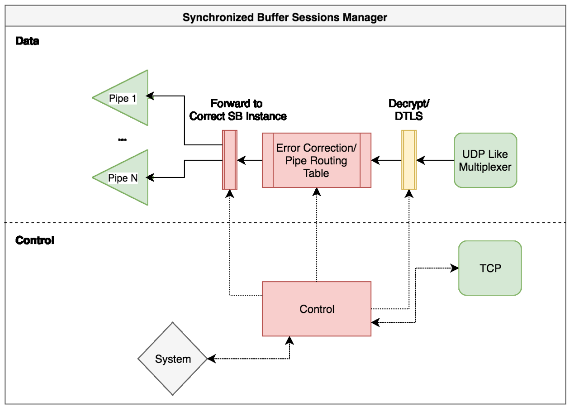

Synchronized Buffer Sessions Manager
Data
Pipe 1
...
Pipe N
Forward to Correct SB Instance
Error Correction/ Pipe Routing Table
Decrypt/ DTLS
UDP Like Multiplexer
Control
TCP
Control
System

Fig. 5. Synchronized Buffer Session Manager

G. Sessions Manager

1) Description: The sessions manager has three primary purposes. The first purpose is to collect, decode using optional forward error correction codes (FECs) [20] and reassemble data frames through a UPD or other connectionless communication protocol. Next data frames de-multiplexed from multiple synchronized buffer sources, and pushed to the correct synchronized buffer instances. Finally a TCP or reliable communication protocol control port routes control messages for each synchronized buffer instance. Additionally the sessions manager should manage compression, especially of video data using a format like H.265 [10].

2) Forwarding and Mechanics: The sessions manager initiates new sessions and verifies the identity of sources. If the data frame has a valid sessions ID, it is forwarded to the correct session by the routing table. If the frame is a control frame and has a valid sessions ID, it is forwarded to the correct control session by the routing table. Finally if the frame is initialization or some other function defined by a specific implementation.

If a session is terminated, the session's control pipe sends a control frame with no data to the sessions manager which then tears down the synchronized buffer instance and passes the frame back to all its senders. Any routes are then removed from the routing table.

3) Multiplexing and Port Connections Manager: Multiplexing allows all transmitters to send all data to a predetermined port or similar abstraction. This allows the system to conserve sockets, especially for receiving streamed data from multiple sources. Note that IP or similar frame sniffing will be required to verify the identify of senders. Additionally, multiplexing may be substituted for a port manager that allocates specific ports for each UDP or connectionless communication protocol (control messages would be received by TCP or a reliable communication protocol service which has source multiplexing.

H. Quality of Service (QoS)

1) FECs and De-Jittering: Additionally forward error correcting codes (FECs) [20] allow streamed data to incorporate redundancies to interpolate original data. However, FECs require additional data to be sent and increase end to end protocol latency due to encoding an decoding.

De-jittering [18] [19] can be important to prevent busy connections from unnecessarily either overflowing the queues or drying queues. Jitter Buffers provide a smoother, more consistent data flow before sampling occurs at the cost of additional latency. Note that the sampling is not the same as de-jittering. While not essential, it would allow a jittery output to be smoothed over, negating highly bursty or variable (with regards to DFPS) given an acceptable ϵ.

I. Network Security

For network security, this protocol requires the communication channel to support authentication, integrity, and confidentiality. To satisfy these requirements, we propose the use of DTLS (1.2+) and TLS (1.3+) for the Data and Control transmissions. TLS is the common encryption protocol over TCP that is used for https connections for providing encryption, authentication, and integrity. DTLS is similar but for providing similar security for UDP connections. We chose TLS / DTLS due it being a commonly used and implemented standard in security for providing authentication of the server (in this case, the transmitter), message integrity and confidentiality, as well as its Anti-Replay support and cipher suites with Forward Secrecy in TLS and DTLS [1] [2], and Reliable Session Establishment, large message size capacity, and Denial-of-Service Countermeasures in DTLS [1]. In our project, this helps prevent many attacks and usability problems that could arise using the protocol.

There are two crucial network security elements to consider when using this protocol. First, DTLS does not hide the length of the messages [1]. This can leak information when sending compressed data. An example of this is sending compressed data from a security camera. If the camera is covered up, it may only send black images, which can be compressed more efficiently than a multi-colored images of the environment. As a result, this could reveal if the camera is covered up or not recording properly. Second, it is possible that all the frames sent are corrupted or dropped.

Trial	Client DFPS	Server DFPS	Lowest Average DFPS	Average Output DFPS
1	30,30,30	27,35,10	29	27
2	30,30,30	30,30,30	30	30
3	30,30,30	60,60,60	30	30
4	30,30,30	28,32,30	30	30
5	30,30,30	30,0,30	20	20
6	30,30,30	21,40,27	29.3	29.3
7	30,30,30	27,15,35	25.7	25.7
8	30,30,30	15,15,15	15	15

TABLE I. DFPS

Trial	Time handshake	Time sync algorithm	Total Time for client to run experiment
1	2.3eˆ -4	1.1eˆ -3	3.5eˆ -3
2	2.2eˆ -4	8.9eˆ -4	3.3eˆ -3
3	2.2eˆ -4	3.3eˆ -3	6.0eˆ -3
4	2.3eˆ -4	8.6eˆ -4	3.4eˆ -3
5	2.2eˆ -4	6.7eˆ -4	3.3eˆ -3
6	2.0eˆ -4	8.7eˆ -4	3.6eˆ -3
7	2.2eˆ -4	8.5eˆ -4	3.2eˆ -3
8	2.3eˆ -4	5.4eˆ -4	2.75eˆ -3

TABLE II. TIME DURATION OF EXPERIMENTS IN SECONDS

VI. EVALUATION

1) Overview: In this paper we evaluated the performance of the time synchronization and data syncing in the protocol, using the bare bones version of the protocol, without the Encryption or Forward Error correction options used. We tested this by generating data within client and server scripts on the same computer, at different DFPS, sent the data over the local connection from the server to a client, and ran the algorithm on the two data streams with different latency allowances and evaluated the outputting tuples per seconds. The test data is generated using a tick length of 0.01 and a specified DFPS of for each second over a three second interval. Times stated are in seconds.

We allow 50 milliseconds of time difference between the two pairs of data for it to be synced. This was chosen experimentally on the premise of being able to adapt to drops and bursts in network over the 3 seconds of data while still allowing for significantly faster decisions from autonomous vehicles then average human reaction times to traffic sign messages while driving, which were at least 540 ms in the study cited [25]. This allows a self driving car system to take up to almost 490 milliseconds to process remote data and react to beat the average human reaction time. Given the existence autonomous driving systems with inference speeds under 100ms [26], we believe this is a large enough of a time interval to be useful for self driving cars.

We also ran an experiment running the time sync algorithm on the local computer 1000 times. The average time difference between the client and server was 1.2eˆ -4 seconds, with a standard deviation of 1.3eˆ -4 seconds and 3 standard deviations above the mean of 5.1eˆ -4 seconds. Thus, given a network channel with equal latency from the client to the transmitter and the transmitter to the client, a time difference between the client and server using the manual time sync method should be under 5eˆ -4 seconds.

2) Analysis: From the first Table's Server DFPS and Average Output DFPS columns, it is shown that the time sync algorithm handles the transmitter sending bursting, reduced DFPS, and up to an extent, missing data. This shows that the Time Matching Algorithm is robust with sufficient latency allowance against these conditions.

It should also be noted that running the experiment took an average of 3.628 milliseconds. With the 5G and C-V2X networks theoretical end to end latency of 1ms [21], and four network connections during the exchange, this should have taken around a maximum average of 5-6 milliseconds with the tested configuration. As long as the data is sufficiently small, a 50 millisecond window to send and match data should be sufficient in most circumstances using the existing configuration.

This also shows that local data frames only have approximately a 3-4 millisecond delay when using the protocol in the tested configuration.

Future experiments will be performed between actual vehicles and infrastructure using efficient Forward Error Correction and TLS/DTLS implementations, as well as introducing other variables such as vehicle speed and C-V2X connection context switching, lowering the allowed time difference between data streams and using three network connections instead of four.

VII. Conclusion

In conclusion, we designed a protocol that permits streaming from n sources to a single destination. By allowing arbitrary data frames per second, we give models the flexibility in relevant streaming data that is synchronized by time-stamps. By building our protocol on top of existing physical and transport layer architecture such as C-V2X, this protocol is able to ignore specific hardware and firmware implementation details, allowing different infrastructure and different vehicle types to communicate universally. This decision also allows network layer or protocol layer middle box optimization software to minimize bandwidth and prevent network saturation. Additionally, by encrypting data across each individual data stream, we prevent information from being eavesdropped or modified during transit.

References

[1] E. Rescorla and N. Modadugu, Datagram Transport Layer Security Version 1.2, IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed: 08-Jan-2018].

[2] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, IETF Tools. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-23. [Accessed: 08-Jan-2018].

[3] J. Lianghai, M. Liu, A. Weinand, H. Schotten Direct Vehicle-to-Vehicle Communication with Infrastructure Assistance in 5G NetworkarXiv Aug 2017

[4] K. Abboud, H. A. Omar and W. Zhuang, "Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey," in IEEE Transactions on Vehicular Technology, vol. 65, no. 12, pp. 9457-9470, Dec. 2016.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars. ArXiv e-prints, Apr. 2016.

[6] MCData Architecture and Flows (Release 15), 3GPP TS 23.282 In-Progress.

[7] A. Papathanassiou and A. Khoryaev, Cellular V2X as the Essential Enabler of Superior Global Connected Transportation Services, IEEE 5G Tech Focus: Volume 1, Number 2, June 2017, 2017.

[8] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 2003.

[9] L. Bariah, D. Shehada, E. Salahat, and C. Y. Yeun, Recent Advances in VANET Security: A Survey, 2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall), 2015.

[10] G. J. Sullivan, J. R. Ohm, W. J. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[11] Ashritha M and Sridhar C S, "RSU based efficient vehicle authentication mechanism for VANETs," 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, 2015, pp. 1-5.

[12] IEEE Std 1003.1-2016 and The Open Group Technical Standard Base Specifications, Issue 7 2016.

[13] S. h. Lee, S. h. Park and S. s. Choi, "Enhanced Synchronization Method Between IEEE1394 and IEEE802.15.3," IEEE Vehicular Technology Conference, Montreal, Que., 2006, pp. 1-4.

[14] B. W. Khoueiry and M. R. Soleymani, An Efficient Noma V2X Communication Scheme in the Internet of Vehicles, 2017 IEEE 85th Vehicular Technology Conference (VTC Spring), 2017.

[15] M. Boban, K. Manolakis, M. Ibrahim, S. Bazzi and W. Xu, "Design aspects for 5G V2X physical layer," 2016 IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, 2016, pp. 1-7.

[16] Timing and Synchronization, IEEE 802.1AS, 2011

[17] D. C. Mazur, R. A. Entzminger, J. A. Kay and P. A. Morell, "Time synchronization mechanisms for the industrial marketplace," 2015 IEEE/IAS 51st Industrial and Commercial Power Systems Technical Conference (ICPS), Calgary, AB, 2015, pp. 1-7.

[18] B. Oklander and M. Sidi, "Jitter Buffer Analysis," 2008 Proceedings of 17th International Conference on Computer Communications and Networks, St. Thomas, US Virgin Islands, 2008, pp. 1-6.

[19] Voznak M., Kovac A., Halas M. (2012) Effective Packet Loss Estimation on VoIP Jitter Buffer. In: Becvar Z., Bestak R., Kencl L. (eds) NETWORKING 2012 Workshops. NETWORKING 2012. Lecture Notes in Computer Science, vol 7291. Springer, Berlin, Heidelberg

[20] Rizzo L. zfec https://pypi.python.org/pypi/zfec

[21] M. I. Ashraf, Chen-Feng Liu, M. Bennis and W. Saad, "Towards low-latency and ultra-reliable vehicle-to-vehicle communication," 2017 European Conference on Networks and Communications (EuCNC), Oulu, 2017, pp. 1-5.

[22] John R. Douceur. 2002. The Sybil Attack. In Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01), Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron (Eds.). Springer-Verlag, London, UK, UK, 251-260.

[23] P. Vyskocil and J. Sebesta, "Relative timing characteristics of GPS timing modules for time synchronization application," 2009 International Workshop on Satellite and Space Communications, Tuscany, 2009, pp. 230-234.

[24] H. C. Berns and R. J. Wilkes, "GPS time synchronization system for K2K," in IEEE Transactions on Nuclear Science, vol. 47, no. 2, pp. 340-343, Apr 2000.

[25] J. ELLS and R. DEiVAR, Rapid Comprehension of Verbal and Symbolic Traffic Sign Messages. 1979.

[26] E. Hou, S. Hornauer, and K. Zipser, Fast Recurrent Fully Convolutional Networks for Direct Perception in Autonomous Driving. 17-Nov.-2017.

[27] A. English, P Ross, D Ball, B Upcroft, P Corke. TriggerSync: A time synchronization tool. IEEE, 2015.

[28] S. Liu, J. Schulze, T. Defanti. Synchronizing Parallel Data Streams via Cross-Stream Coding