
Real-Time LiDAR Data Transmission to the Edge over mmWave

Kevin Hermstein

Gil Zussman

Electrical Engineering, Columbia University, New York, NY

K.HERMSTEIN@COLUMBIA.EDU

GIL.ZUSSMAN@COLUMBIA.EDU

Abstract

We present a GNU Radio out-of-tree (OOT) module for transmitting, receiving, and processing LiDAR data in real-time. The transmitter packages the data broadcasted over Ethernet from a Velodyne LiDAR in an OFDM packet. The receiver handles packet detection, alignment, carrier frequency and phase offset correction, and equalization. Finally, the demodulated packed LiDAR payload is converted to a standard XYZ format for downstream processing, such as visualization in a live 3D point cloud. The module is tested over the air on a mmWave link using 28GHz phased array antennas at the transmitter and receiver, along with USRP software-defined radios. The system is tested in a controlled indoor environment, as well as outdoors on the NSF COSMOS testbed in NYC.

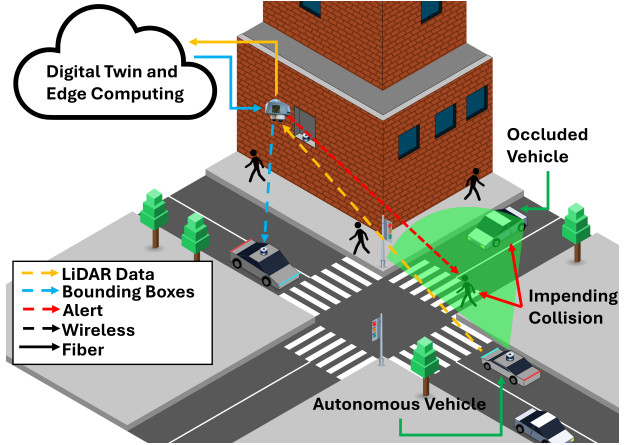


Figure 1. DT data collection and alert system for an urban intersection. LiDAR data collected by AVs are transmitted to the edge and processed at the edge-cloud. Instructions and alerts are then sent to end users.

1. Introduction

Autonomous vehicles (AVs) hold the potential to create safer and more efficient cities and roads. However, they are encumbered with high bandwidth sensor data which requires heavy computation, along with strict latency requirements. These challenges are magnified when considering cooperative vehicles, in which data is shared between vehicles, extending effective sensing range and revealing otherwise occluded obstacles. There has been work on vehicle-to-vehicle (V2V) and vehicle-to-everything (V2X) infrastructure to accommodate the high bandwidth demand of data from multiple sensors. Existing works explore different scenarios where sensors are infrastructure-based (Arnold et al., 2022), or deployed on AVs (Wang et al., 2020; Chen et al., 2019). Across these works, the LiDAR data is encoded to distill the data and lighten the bandwidth requirement at the cost of some sensing performance.

Additionally, digital twins (DTs) offer promising capabilities in improving pedestrian safety by utilizing camera, Li-

DAR, and radar-based sensing. Previous works on smart cities have utilized real-time video processing (Ghasemi et al., 2025; Fu et al., 2026). However, utilizing cameras introduces concerns with privacy, whereas radar and LiDAR preserve anonymity.

We envision a system pictured in Figure 1, where AVs continuously offload their raw sensor data to edge-cloud servers. Processing is then performed at the edge-cloud. The sensing data, such as bounding boxes, can be sent to vehicles and pedestrians or can be further processed into commands or alerts, similar to (Ghasemi et al., 2025). In addition to performing sensing, the edge-cloud server can also maintain a real-time DT of the environment. At a local level, this can enable high resolution localization to enhance safety. Multiple edge servers can then send this data to a central controller which can maintain a city-scale DT, enabling city-wide optimization of traffic patterns as well as the collection of road and pedestrian traffic data, all while preserving privacy.

All of these benefits can only be realized by addressing the communication bottleneck. We take the first step towards this by developing *gr-LiDAR*, an out-of-tree module for GNU Radio for transmitting, receiving, and processing Li-

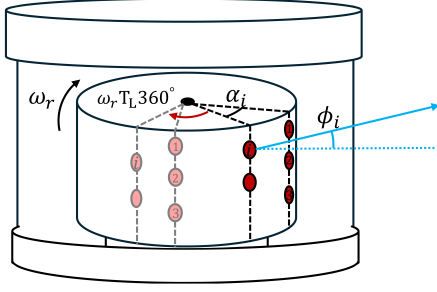


Figure 2. LiDAR operation. The lasers are not aligned in azimuth and the sensor mechanically rotates.

DAR data in real time. The transmitter packages the incoming LiDAR data in an orthogonal frequency division multiplexing (OFDM) packet in real time. The receiver handles packet detection, alignment, carrier frequency offset (CFO) correction, phase offset correction, and equalization. Additionally, we present a block to convert the LiDAR packet payload into standard XYZ format for downstream processing. We evaluate the communication abilities of the system in terms of latency. We also provide a Python application for live 3D point cloud plotting, and an optimized pre-trained model for real-time object detection.

Finally, we present a full end-to-end system with a Velodyne LiDAR, USRP software-defined radios (SDRs), and 28 GHz mmWave phased array antenna modules (PAAMs) (Sadhu et al., 2017; Gu et al., 2019; Maddala et al., 2024; Gu et al., 2021). This system is verified in controlled laboratory environments as well as outdoors on the COSMOS testbed in NYC (Raychaudhuri et al., 2020).

2. System Overview

2.1. LiDAR Operation

Generally, a LiDAR operates with N_L lasers. The i^{th} laser has an associated elevation angle ϕ_i and azimuthal angle offset α_i ($i = 1, \dots, N_L$). The LiDAR rotates mechanically and fires all lasers every T_L seconds. This operation is illustrated in Figure 2. In a full 360° rotation there are $N_s = \frac{\omega_r T_L}{360^\circ}$ firings, where $\omega_r = \frac{1}{T_L}$ and ω_r is the rotations per second of the LiDAR. Because T_L is fixed, the data rate of the LiDAR is fixed. There are $\omega_r \cdot 360^\circ \cdot T_L$ degrees of separation between each firing. Increasing ω_r leads to larger angular separation of the data, but less time between illuminations of the same area. This inherently leads to a tradeoff of sensor resolution and refresh rate of the data.

Each laser produces B_L bytes of data to encode distance and intensity. Because the data per laser is small, we assume the LiDAR packages all N_L lasers together in one data block, along with B_o overhead bytes. Overhead bytes can account for timestamps, motor read backs, configura-

tions, or other flags. A single data block is $B_L N_L + B_o$ bytes. The LiDAR then packages N_B blocks together, along with additional header bytes, B_h for the data transport and configuration data. The total packet size in bytes is then

$$B_{packet} = N_B(B_L N_L + B_o) + B_h \quad (1)$$

2.2. Communication Waveform

We assume the LiDAR is deployed on an autonomous vehicle connected to the edge over a cellular connection using OFDM. The OFDM waveform utilizes N_c subcarriers, with N_d data carriers, each using QAM with modulation order M , N_p pilot subcarriers, and N_n null carriers. A cyclic prefix of length N_{cp} is prepended to each OFDM packet. One LiDAR packet requires $\lceil (8B_{packet}) / (MN_d) \rceil$ OFDM packets to transmit. We assume these OFDM packets are all assembled in the same frame. In addition to the data packets, there are N_{pp} pilot packets. The whole OFDM frame length in samples is then

$$L = (N_c + N_{cp}) \left(\left\lceil \frac{8 \cdot B_{packet}}{MN_d} \right\rceil + N_{pp} \right) \quad (2)$$

2.3. Base station

The base station (BS) is deployed near an edge-cloud deployment which contains CPUs and GPUs. The BS detects, synchronizes, and demodulates the incoming OFDM packets. Once the LiDAR packets is demodulated, the packed data is converted to XYZ standard point cloud data. This data can then be sent over fiber and processed for object detection on edge-cloud servers. The edge-cloud returns bounding boxes and other lightweight information back to the BS for transmission to end users.

2.4. Latency Requirements

Pedestrians must be warned with 1-2 seconds to react in order to provided enough time to avoid collision (Ghasemi et al., 2025). This sets a strict latency requirement for the system, dependent on the approaching vehicle's speed and distance to the pedestrian. The total latency of the system is

$$\tau = \tau_{fire} + \tau_{tx,up} + 2\tau_{prop} + \tau_{rx} + \tau_{detect} + \tau_{tx,down}$$

Each term that makes up the total latency is described below:

- τ_{fire} : Inherent delay in firing lasers of the LiDAR and outputting the data. This is a constant associated with the LiDAR.
- $\tau_{tx,up}$: Time to construct the OFDM frame and transmit the final waveform.

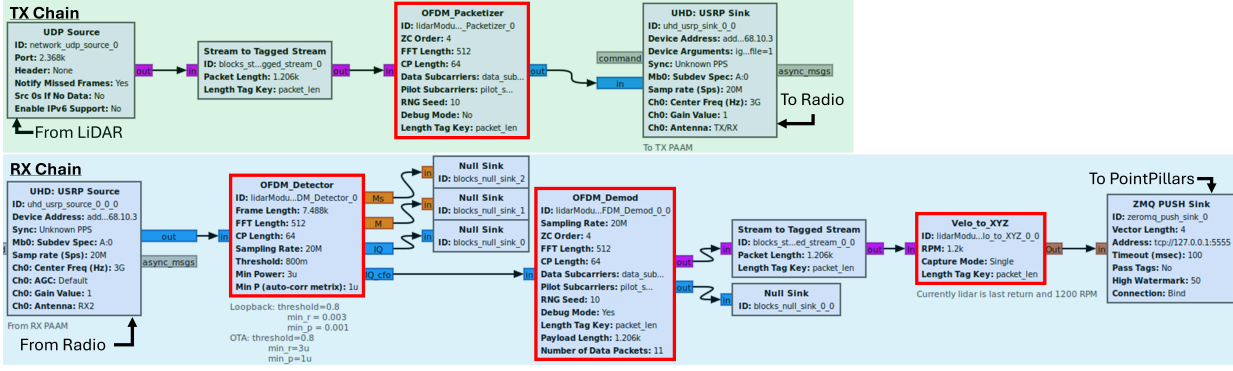


Figure 3. Top (green) the full Tx flowgraph. Bottom (blue) the full Rx flowgraph. The red highlighted blocks are from `gr-LiDAR`.

- τ_{prop} : The propagation delay, d/c , where c is speed of light and d is the distance from the AV to the BS. Doubled to account for the downlink transmission.
- τ_{rx} : Time to detect and demodulate the OFDM frame.
- τ_{detect} : Time to perform object detection on the received LiDAR data.
- $\tau_{tx,down}$: Time to transmit the bounding boxes or commands to end users.

The time to transport the LiDAR data to the edge-cloud compute server is negligible as it is co-located with the BS and connected over fiber.

3. `gr-LiDAR`

In this section we introduce `gr-LiDAR`, an OOT module for transmitting, receiving, and processing LiDAR data in GNU Radio. It is important to note, that although the purpose of this work is for applications of real-time edge-cloud processing of LiDAR data, any data can be modulated onto the OFDM waveform. The OOT module is designed for real-time processing, utilizing libraries such as Eigen (Eigen, 2026), Volk (Volk, 2020), and FFTW (FFTW, 2014) for efficient signal processing. A complete flowgraph utilizing all `gr-LiDAR` blocks is shown in Figure 3.

FFT and IFFT

The IFFT and FFT operations are vital to OFDM packet creation and demodulation. We utilize the FFTW library (Frigo & Johnson, 2005; FFTW, 2014) which learns the most efficient way to numerically calculate the DFT on any hardware system. The entries of the input/output buffers correspond to the OFDM subcarriers. FFTW buffers can also maintain alignment for SIMD commands, enabling efficient downstream processing with no memory copies.

OFDM Packetizer

The OFDM Packetizer block handles modulating the incoming LiDAR data onto the subcarriers of the OFDM waveform, prepending the cyclic prefix, and assembling the final frame with pilot packets. The packetizer is flexible, allowing user specified subcarrier allocations, FFT lengths, and CP lengths.

There are two pilot packets in the final waveform, one for detection and alignment, and one for channel estimation. These are analogous to the primary synchronization signal (PSS) and secondary synchronization signal (SSS) in 5G-NR. Both the PSS and SSS are truncated Zadoff-Chu (ZC) sequences generated by first assigning the sequence to each subcarrier and then taking the IFFT and prepending the cyclic prefix (Frank et al., 1962; Andrews, 2026). The detection pilot contains a repeated sequence of $N_c/2$ length ZC pilots to enable autocorrelation based detection. The BPSK comb pilot symbols are normal randomly distributed.

All of the pilot packets and symbols are generated in the class constructor. This avoids unnecessary compute time of generating the pilots for every frame. A byte stream with a length tag is expected as the input. The output is a complex valued tagged stream, the complete OFDM frame. The output frame length is deterministic, given by Equation 2.

For each tagged sequence of incoming bytes, the following is done. First, the precomputed PSS and SSS are copied to the output buffer. The comb pilots are copied into the appropriate indices of the IFFT input buffer only a single time. All of the symbols for the incoming byte sequence are generated. Then N_d symbols are taken and copied onto the appropriate indices of the IFFT input buffer. The IFFT is taken, the CP is then prepended to the packet and the packet is copied to the output buffer. This is repeated until all symbols are copied and the frame is complete.

OFDM Detector

The detection and synchronization algorithm utilized is the

autocorrelation-based Schmidl-Cox algorithm (Schmidl & Cox, 1997). The general principle is to perform an autocorrelation on the incoming samples with an offset of $L_o = N_c/2$. The autocorrelation is calculated as $P(d) = \sum_{n=0}^{L_o-1} r_{d+n} r_{d+n+L_o}^*$, for the d^{th} sample, where r_n is the n^{th} sample of the incoming signal. In practice, this is calculated recursively as

$$P(d+1) = P(d) + r_{d+L_o} r_{d+2L_o}^* - r_d r_{d+L_o}^*$$

The final metric, $M(d)$, is $P(d)$ scaled by the energy of the received samples, $R(d)$, giving $M(d) = \frac{|P(d)|^2}{(R(d))^2}$. In implementation, $P(d)$, $R(d)$, and $M(d)$ are circular buffers of size N_c . When $R(d)$ is small, the floating-point division for calculating $M(d)$ can cause large errors, thus we also implement an initial power-gating check.

The $M(d)$ metric is maximized at the start of the PSS, and maintains this value for the duration of the CP. To find a defined start of the packet, we perform a moving average of the N_{cp} most recent samples. This results in a peak at the end of the CP. We detect the peak as the max averaged value whose following value decreases.

Once this sample is found, the block then performs coarse CFO correction. For a CFO of Δf , there is a residual phase offset of $\phi = 2\pi T \Delta f$ between the samples, where $T = \frac{N_c}{2f_s}$ is the time between identical samples and f_s is the sample rate. Because $r_d = r_{d+L_o}$, multiplying $r_d r_{d+L_o}^*$ removes the phase effect of the channel and isolates the residual phase caused by the CFO. Thus, $\angle P(d)$ captures the residual phase. The CFO can then be recovered as

$$\Delta \hat{f} = \frac{\angle P(d)}{2\pi T} = \frac{\angle P(d) f_s}{\pi N_c}$$

Now that the start of the frame is found and the CFO is estimated, the samples are pushed downstream with a corrective phase adjustment of $\exp(-2\pi n \Delta \hat{f} / f_s)$ for the n^{th} sample. The start of the stream is tagged with the frame length.

OFDM Demod

The OFDM Demod block takes the coarse frequency corrected samples from the OFDM Detector. It handles channel estimation, equalization, residual phase correction, and demodulation. Least-squares channel estimation is used, $\hat{H}[f_n] = \frac{Y[f_n]}{X[f_n]}$. The corrected samples are then calculated as $W[f_n] = \frac{Y[f_n]}{\hat{H}[f_n]}$. For each packet we also analyze the residual phase in the pilots and the DC offset. The phase offset is calculated as $\hat{\phi} = \frac{1}{N_p} \sum_{k=0}^{N_p-1} \angle (W[f_k] / X[f_k])$, where f_k is the k^{th} pilot subcarrier. Each symbol is then corrected by multiplying by $\exp(-j\hat{\phi})$. A similar process is done to estimate the DC offset, where we find the complex mean of the data and subtract it from each symbol.

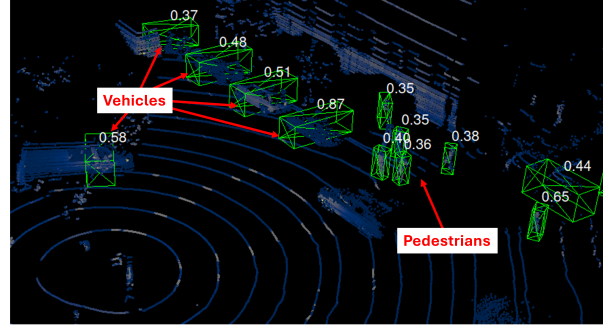


Figure 4. An example of an output frame of the system. Vehicles and pedestrians are detected and the bounding boxes are rendered in the point cloud in real-time. This scene is a busy city street with parked vehicles, moving vehicles, and many pedestrians which is especially challenging for the model.

Variable	Value
Header (B_h)	48
Blocks (N_B)	12
Overhead (B_o)	4
Lasers (N_L)	32
Laser Data (B_L)	3
B_{packet} (Eq. 1)	1248

Table 1. Breakdown of the LiDAR data packet structure.

Demodulation is then performed on these samples and the bytes are assembled and pushed downstream. To ensure efficient data processing, the channel estimation and correction utilizes Eigen to perform SIMD operation on the occupied carriers. Additionally, the output memory buffer of the FFTW plan is also mapped to an Eigen vector, removing the need for additional memory copies. The only memory copies needed are to move the input stream to the FFTW input buffer, and to copy bytes to the output buffer.

Velo to XYZ

At this point, the LiDAR packet has been received, however it is stored in a format conducive to communication, not one that can be interpreted by existing models to perform object detection. This block converts the LiDAR packet into XYZ format: A four element vector that describes the X, Y, Z coordinates, and the intensity of a single point. Details of the LiDAR packet structure depends on the LiDAR model. In our case, we use the VLP-32c LiDAR. We provide an overview and highlight the nuances to ensure accurate data retrieval. Further details can be found in the manual (Lidar, 2021).

Table 1 shows the breakdown of the LiDAR data packet structure. The Velodyne LiDARs utilize UDP, and thus contain a 42-byte UDP header, along with a four-byte timestamp and two factory bytes at the end of the packet which we count as header bytes. Following the header, 12

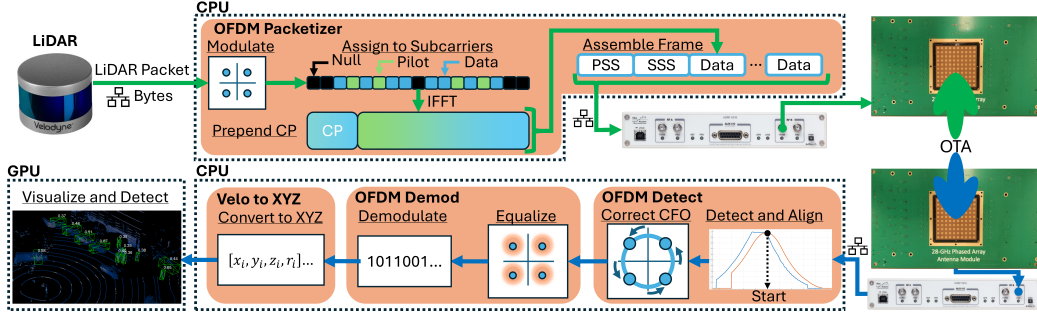


Figure 5. Full system diagram. Portions in orange boxes are individual GR blocks.

data blocks contain the actual LiDAR data. Each block contains four overhead bytes, a two-byte flag and a two-byte azimuth reading, and data from 32 lasers. Each data point is three bytes, two bytes of range data, and one byte of reflectivity. The order of firings in the data block are specified in the manual. In total, the packet is 1248 bytes.

For each data block, we can recover the azimuth angle, α_d , and each distance d_i for all N_L lasers. Using the known elevation angle ϕ_i for the i^{th} laser, the cartesian coordinates relative to the LiDAR are

$$\begin{aligned} x_i &= d_i \cos(\phi_i) \sin(\alpha_d) \\ y_i &= d_i \cos(\phi_i) \cos(\alpha_d) \\ z_i &= d_i \sin(\phi_i) \end{aligned}$$

This however, neglects two things: 1) The azimuthal offset of each laser. 2) The drift of the LiDAR rotation between individual laser firings shown in Figure 2. Issue 1 is easily accounted for by utilizing the offset listed in the manual. Issue 2 requires utilizing the specified time between consecutive laser firings, t_{fire} . The drift between consecutive firings is then $\delta = t_{fire}\omega_r$. However, the motor can often drift from the designated ω_r , thus we estimate ω_r from consecutive azimuth readings as $\hat{\omega}_r = (\alpha_d^j - \alpha_d^{j-1})/T_L$, for the j^{th} data block. For the last data block we reuse the previous $\hat{\omega}_r$. The XYZ coordinates are more accurately

$$\hat{x}_i = d_i \cos(\phi_i) \sin\left(\alpha_d - \alpha_i + \left\lfloor \frac{i}{2} \right\rfloor \hat{\delta}\right) \quad (3)$$

$$\hat{y}_i = d_i \cos(\phi_i) \cos\left(\alpha_d - \alpha_i + \left\lfloor \frac{i}{2} \right\rfloor \hat{\delta}\right) \quad (4)$$

$$\hat{z}_i = d_i \sin(\phi_i) \quad (5)$$

where i denotes the laser ID of the data point within a single data block and $\hat{\delta} = t_{fire}\hat{\omega}_r$. In the case of the VLP-32c LiDAR, the lasers are fired in pairs, thus we only accumulate the drift every two data points. The final vector output by the block is then: $[\hat{x}_i \ \hat{y}_i \ \hat{z}_i \ r_i]^T$ where r is the reflectivity. This vector stream can then be readily interpreted directly by models or visualized as a point cloud, as shown in Figure 4.

4. Testbed

The OOT module is verified on the COSMOS testbed in NYC (Raychaudhuri et al., 2020). The COSMOS testbed is built to support experimentation with next generation networks, especially in areas such as mmWave, full-duplex, edge-computing, DTs, optical x-haul, and quantum networks. We utilize the 28 GHz phased array antenna modules (PAAMs) (Sadhu et al., 2017; Gu et al., 2019; Chen et al., 2023) along with USRP SDRs. The full system is illustrated in Figure 5, with the gr-LiDAR blocks highlighted in orange.

To emulate an AV, we utilize the mobile PAAMs (Madala et al., 2024), which packages the PAAM frontend and USRP-2974 SDR together. The USRP-2974 contains an X310 SDR and an embedded Intel i7-6822 CPU, combining the host and SDR. The LiDAR is co-located and connected to the CPU over Ethernet. The transmit flowgraph, shown in the green part of Figure 3, utilizes a UDP Source to receive the LiDAR data. This data is then modulated onto the final OFDM waveform, which is transmitted over the air. The top of Figure 5 shows the full TX system.

The BS is emulated with a similar setup. Again, a PAAM frontend and a USRP-2974 are utilized, however the on-board CPU is not utilized. We utilize a host PC with a Intel i9-13900K CPU and a GeForce GTX 1080 GPU. The receive flowgraph, shown in the blue part of Figure 3, reads samples from the SDR, then performs detection and demodulation. The recovered LiDAR packets are then converted to XYZ format. The bottom of Figure 5 shows the full RX system.

We show two different applications, visualization and object detection. First, we show real time visualization of the LiDAR point cloud. Because GNURadio is multi-threaded, the 3D point cloud visualizer is implemented as a separate process. The XYZ data is passed to this separate process using ZMQ. The visualizer rapidly updates a table of the most recent points and re plots this data at a specified frame rate utilizing PyQt and OpenGL.

For object detection, we utilize PointPillars (Lang et al., 2019). We adapt CUDA-PointPillars (Lei Fan, 2022) to run in real time on a GTX-1080. We use model weights pretrained on the open source KITTI dataset (Geiger et al., 2013). These weights are optimized with TensorRT for real-time performance (NVIDIA, 2026). The underlying modified CUDA-PointPillars program C++ API which is invoked by a Python script through the ctypes library (Python Software Foundation, 2026). The API only requires the memory address of a Numpy array to operate, removing the latency from memory copies.

4.1. Performance

We characterize the latency of each block in `gr-LiDAR` along with the object detection model. We run the testbed setup in an indoor lab environment for 72 seconds. The OFDM Detect block introduces a constant delay of N_{cp}/f_s to accumulate samples for $M(d)$ and begin passing on corrected samples. In our system $N_{cp} = 64$ and $f_s = 20$ MHz leading to a latency of 0.003 ms. All other latencies are measured empirically.

The full results are summarized in Table 2. We observe the `gr-LiDAR` blocks are responsible for a negligible portion of the processing latency (0.8%). The majority of the latency comes from the object detection with PointPillars. We note this can be accelerated on more powerful GPU platforms as the GTX-1080 is outdated. This still illustrates the feasibility in providing real-time data to other edge devices.

In the 72-second experiment, 93,967 LiDAR packets are processed by the transmitter. This implies a packet rate of 1305 packets per second, just shy of the expected packet rate of the LiDAR, which can be attributed to the network interface dropping some packets. Of the transmitted packets, 93,944 packets are detected by the receiver (99.98%). Across 93,944 packets there are 1,127,328 data blocks, of which 4,124 contain errors, preventing proper recovery of the LiDAR. This is a data block error rate of 0.37% (Note this is not the typical block error rate in FEC. This is the error rate of the data blocks in the received LiDAR packets). Overall, we can observe that in a controlled setting the communication performance is both efficient in both runtime and accuracy.

We qualitatively can observe the performance of the real-time PointPillars model in Figure 4. We can see that in the challenging setting of a busy urban intersection, the model is able to detect numerous vehicles and pedestrians in real-time.

A video of the communication performance on the COSMOS testbed is available (Hermstein, 2026). This shows the real-time transmission and point cloud visualization

Block	Hardware	latency (ms)
OFDM Packetizer	i7-6822	0.065
OFDM Detect	i9-13900K	0.003
OFDM Demod	i9-13900K	0.039
Velo2XYZ	i9-13900K	0.009
PointPillars	GTX-1080	13.48
Total		13.59

Table 2. Latency introduced from each portion of the system.

aligned with camera video on the COSMOS testbed, both indoors and outdoors. The video also shows the emulated performance by replaying pcaps of LiDAR data to the transmitter which is the transmitted over the air in a controlled environment. At the receiver the real-time object detection model can successfully detect vehicles, pedestrians, and cyclists.

5. Conclusion

In conclusion, this work presents a real-time transmitter and receiver to support real world experimentation with cooperative sensing and DTs. We discuss the practical nuances of implementing a real-time OFDM transceiver and retrieving accurate LiDAR data. We successfully implement a full system on the COSMOS testbed utilizing mmWave frontends and USRP SDRs to offload the raw LiDAR sensor data to the edge-cloud. By running an optimized pre-trained model on a GPU, we perform object detection on the transmitted data in real-time. The system is benchmarked in terms of latency and error rate.

6. Acknowledgments

This work was supported in part by NSF Center for Smart Streetscapes (CS3) under NSF Cooperative Agreement EEC-2133516, NSF grant CNS-2148128 and by funds from federal agency and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program, NSF grants AST-2232455 and CNS-2450567, and by the Air Force Office of Scientific Research under award number FA9550-23-F-0014.

References

- Andrews, Jeffrey G. A primer on Zadoff-Chu sequences, 2026. URL <https://arxiv.org/abs/2211.05702>.
- Arnold, Eduardo, Dianati, Mehrdad, de Temple, Robert, and Fallah, Saber. Cooperative perception for 3D object detection in driving scenarios using infrastructure sensors. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):1852–1864, 2022.

- Chen, Qi, Ma, Xu, Tang, Sihai, Guo, Jingda, Yang, Qing, and Fu, Song. F-cooper: feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC '19*, pp. 88–100, New York, NY, USA, 2019. Association for Computing Machinery.
- Chen, Tingjun, Maddala, Prasanthi, Skrimponis, Panagiotis, Kolodziejewski, Jakub, Adhikari, Abhishek, Hu, Hang, Gao, Zhihui, Paidimarri, Arun, Valdes-Garcia, Alberto, Lee, Myung, Rangan, Sundee, Zussman, Gil, and Seskar, Ivan. Open-access millimeter-wave software-defined radios in the PAWR COSMOS testbed: Design, deployment, and experimentation. *Comput. Netw.*, 234 (C), 2023. ISSN 1389-1286.
- Eigen. Eigen, 2026. URL <https://libeigen.gitlab.io/>.
- FFTW. Fftw, 2014. URL <https://www.fftw.org/>.
- Frank, R., Zadoff, S., and Heimiller, R. Phase shift pulse codes with good periodic correlation properties (corresp.). *IRE Transactions on Information Theory*, 8(6): 381–382, 1962.
- Frigo, Matteo and Johnson, Steven G. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93 (2):216–231, 2005. Special issue on “Program Generation, Optimization, and Platform Adaptation”.
- Fu, Yongjie, Gao, Qi, Dehkordi, Mahshid Ghasemi, Zussman, Gil, and Di, Xuan. Multi-pedestrian safety warning at urban intersections use case of digital twin, 2026. URL <https://arxiv.org/abs/2605.18823>.
- Geiger, Andreas, Lenz, Philip, Stiller, Christoph, and Urtasun, Raquel. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- Ghasemi, Mahshid, Fu, Yongjie, Ouyang, Xinyu, Wang, Peiran, Turkcan, Mehmet Kerem, Tavori, Jhonatan, Kleisarchaki, Sofia, Calmant, Thomas, Grgen, Levent, Kostic, Zoran, Di, Xuan Sharon, Zussman, Gil, and Ghaderi, Javad. Real-time video analytics for urban safety: Deployment over edge and end devices. In *Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing, SEC '25*, 2025.
- Gu, Xiaoxiong, Liu, Duixian, Baks, Christian, Tagerman, Ola, Sadhu, Bodhisatwa, Hallin, Joakim, Rexberg, Leonard, Parida, Pritish, Kwark, Young, and Valdes-Garcia, Alberto. Development, implementation, and characterization of a 64-element dual-polarized phased-array antenna module for 28-GHz high-speed data communications. *IEEE Trans. Microw. Theory Tech.*, 67(7): 2975–2984, 2019.
- Gu, Xiaoxiong, Paidimarri, Arun, Sadhu, Bodhisatwa, Baks, Christian, Lukashov, Stanislav, Yeck, Mark, Kwark, Young, Chen, Tingjun, Zussman, Gil, Seskar, Ivan, and Valdes-Garcia, Alberto. Development of a compact 28-GHz software-defined phased array for a city-scale wireless research testbed. In *IMS'21*, 2021.
- Hermstein, Kevin. Cosmos lidar transmission to the edge over mmwave, 2026. URL <https://youtu.be/xDvC7DUvTg8>.
- Lang, Alex H., Vora, Sourabh, Caesar, Holger, Zhou, Lubing, Yang, Jiong, and Beijbom, Oscar. PointPillars: Fast encoders for object detection from point clouds, 2019.
- Lei Fan, Lily Li. Cuda-pointpillars, 2022. URL <https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars>.
- Lidar, Velodyne. *VLP-32c User Manual*, 2021.
- Maddala, Prasanthi, Kolodziejewski, Jakub, Adhikari, Abhishek, Hermstein, Kevin, Chen, David, Zhu, Liao, Chen, Tingjun, Seskar, Ivan, and Zussman, Gil. Demo: Experimentation with mobile 28 GHz phased array antenna modules. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom'24*, New York, NY, USA, 2024. Association for Computing Machinery.
- NVIDIA, 2026. URL <https://developer.nvidia.com/tensorrt>.
- Python Software Foundation. ctypes-a foreign function library for python, 2026. URL <https://docs.python.org/3/library/ctypes.html>.
- Raychaudhuri, Dipankar, Seskar, Ivan, Zussman, Gil, Korakis, Thanasis, Kilper, Dan, Chen, Tingjun, Kolodziejewski, Jakub, Sherman, Michael, Kostic, Zoran, Gu, Xiaoxiong, Krishnaswamy, Harish, Maheshwari, Sumit, Skrimponis, Panagiotis, and Gutterman, Craig. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. ACM MobiCom'20*, 2020.
- Sadhu, Bodhisatwa, Tousi, Yahya, Hallin, Joakim, Sahl, Stefan, Reynolds, Scott K., Renstrm, rjan, Sjgren, Kristoffer, Haapalahti, Olov, Mazor, Nadav, Bokinge, Bo, Weibull, Gustaf, Bengtsson, Hkan, Carlinger, Anders, Westesson, Eric, Thillberg, Jan-Erik, Rexberg, Leonard, Yeck, Mark, Gu, Xiaoxiong, Ferriss, Mark, Liu, Duixian, Friedman, Daniel, and Valdes-Garcia,

Alberto. A 28-GHz 32-element trx phased-array ic with concurrent dual-polarized operation and orthogonal phase and gain control for 5G communications. *IEEE J. Solid-State Circuits*, 52(12):3373–3391, 2017.

Schmidl, T.M. and Cox, D.C. Robust frequency and timing synchronization for OFDM. *IEEE Transactions on Communications*, 45(12):1613–1621, 1997.

Volk. Volk, 2020. URL <https://www.libvolk.org/>.

Wang, Tsun-Hsuan, Manivasagam, Sivabalan, Liang, Ming, Yang, Bin, Zeng, Wenyuan, and Urtasun, Raquel. V2vnet: Vehicle-to-vehicle communication for joint perception and prediction. In *Computer Vision – ECCV 2020*, pp. 605–621. Springer International Publishing, 2020.