Scalable Distributed Massive MIMO Baseband Processing

Junzhi Gong, *Harvard University;* Anuj Kalia, *Microsoft;*
Minlan Yu, *Harvard University*

https://www.usenix.org/conference/nsdi23/presentation/gong

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the
20th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Scalable Distributed Massive MIMO Baseband Processing

Junzhi Gong
Harvard University

Anuj Kalia
Microsoft

Minlan Yu
Harvard University

Abstract

Massive MIMO (multiple-in multiple-out) is a key wireless technique to get higher bandwidth in modern mobile networks such as 5G. The large amount of computation required for massive MIMO baseband processing poses a challenge to the ongoing softwarization of radio access networks (RAN), in which mobile network operators are replacing specialized baseband processing chips with commodity servers. Existing software-based systems for massive MIMO fail to scale to increasingly larger MIMO dimensions with an ever-increasing number of antennas and users. This paper presents a new scalable distributed system called Hydra, designed to parallelize massive MIMO baseband processing while minimizing the overhead of distributing computation over multiple machines. Hydra's high scalability comes from reducing inter-server and inter-core communication at different stages of baseband processing. To do so, among other techniques, we take advantage of hardware features in modern commodity radios in novel ways. Our evaluation shows that Hydra can support over four times larger MIMO configurations than prior state-of-the-art systems, handling for the first time, 150×32 massive MIMO with three servers.

1 Introduction

Massive MIMO is a key wireless technique to increase spectral efficiency in modern mobile networks such as 5G. Massive MIMO refers to using a large number of radio antennas to simultaneously serve a large number of users on the same frequency resources. Mobile network operators today are deploying multi-user massive MIMO to handle the increasing demand from mobile users [16]. For example, T-Mobile recently demonstrated the benefits of massive MIMO in a setup with 64 antennas serving eight concurrent users [12], achieving an impressively high total downlink bandwidth of 5.6 Gbps. A promising way to handle the demand for higher spectral efficiency and mobile bandwidth is to increase the massive MIMO dimensions: the number of radio antennas, and the number of users served simultaneously [17, 26, 28].

While the previous-generation LTE networks typically used small MIMO configurations (e.g., four antennas), massive MIMO deployments with 64 antennas are already commonplace in 5G, and future deployments could use hundreds of antennas [16]. For example, AirSpan's Air5G 7200 already supports 128 transmit and 128 receive antennas [2].

This paper tackles the challenge of scalably supporting increasing massive MIMO dimensions in *virtualized* RANs (vRAN). With vRANs, mobile network operators are replacing specialized RAN hardware, such as ASICs and DSPs for wireless signal processing, with commodity x86 servers [5, 10, 11, 13, 15]. RAN virtualization offers important benefits, such as mitigating vendor lock in and increasing RAN flexibility and feature velocity. However, massive MIMO remains a challenge for software-based RANs [8]. This is due to the extremely high computational requirements of massive MIMO, in the presence of tight millisecond-scale latency deadlines. For example, the largest massive MIMO configuration considered in this paper—150 antennas and 32 users—requires our system (Hydra) to use 71 CPU cores, cumulatively handling 80.6 Gbps of fronthaul traffic, within a latency deadline of 2.5 ms.

Our goal is to design a system that can efficiently scale to increasing massive MIMO dimensions by using the resources of more servers, to handle the requirements of 5G and future radio technologies. Key to Hydra's scalability is a set of new techniques that we design to scalably distribute massive MIMO computation among a pool of servers while minimizing the distribution overhead from inter-server and inter-core communication. Existing projects that implement massive MIMO baseband processing in software, such as Agora [17] and BigStation [28], lack a path for scaling to increasing MIMO dimensions. One the one hand, single-machine systems like Agora and Intel's FlexRAN [3] are limited to the CPU and network bandwidth resources of only one machine. One the other hand, the BigStation project studies the opportunities for distributing multi-user MIMO computation, but does not seek to optimize the distribution overhead, which is the focus of this work.

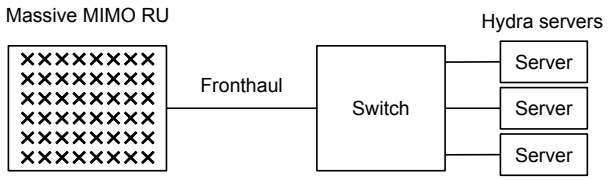


Figure 1: The architecture of a virtualized and distributed massive MIMO baseband processing system.

A massive MIMO baseband processing system (also called a baseband unit, or BBU) connects to a multi-antenna radio unit (RU) over a wired fronthaul link (Figure 1). In our setup, the BBU consists of one or more servers in a datacenter, connected to the fronthaul via an Ethernet switch. The RU and the BBU exchange packets containing in-band and quadrature (IQ) samples for hundreds or thousands of Orthogonal Frequency Division Multiplexing (OFDM) subcarriers. The BBU's computation consists of a pipeline of stages, each with a different type of parallelism [18, 28]. Hydra's design includes three key new ideas (summarized below) to map these BBU computation stages and the inter-stage shuffling of intermediate data to different hardware components, while minimizing inter-server and intra-server communication for scalability.

Taking the uplink direction as an example, the computation stages are as follows (Figure 2). First, antenna-parallel processing converts each antenna's time-domain IQ samples into the frequency domain using a fast Fourier transform (FFT). Second, subcarrier-parallel processing converts each subcarrier's per-antenna IQ sample streams into demodulated per-user streams. Third, user-parallel processing runs forward error correction on the per-user streams, converting them to user bit streams. The BBU connects these stages using communication mechanisms that shuffle the outputs of one stage into the inputs of the next stage.

Our first two ideas reduce inter-server communication (compared to BigStation), and the third reduces intra-server communication (compared to Agora).

1. **BBU-RU interface.** We identify that an existing hardware feature in modern RUs—the ability to perform FFT and generate separate packets for configurable subcarrier ranges—can be used to build a scalable distributed system for massive MIMO. Note that these RU abilities were not originally intended to build scalable distributed systems (Section 3.1); instead, we found a novel use case of these abilities. These FFT-capable RUs exchange frequency-domain IQ samples with the BBU, instead of time-domain IQ samples like in BigStation. Hydra routes packets for different subcarrier ranges to different servers, partitioning the fronthaul traffic and feeding the subcarrier-parallel pipeline stage with near-zero overhead. Compared to BigStation's design in which BBU servers run FFT and shuffle subcarrier ranges in software, our approach reduces inter-server communication by up to 66.4%.

2. **Within the BBU cluster.** Due to abundant parallelism, massive MIMO BBU processing offers many options to distribute computation within the server pool, at the cost of inter-server communication. For example, BigStation shuffles the BBU's intermediate data over the network within the subcarrier-parallel stage, and predicts benefits from splitting individual matrix inverse operations across servers. Such inter-server communication limits BBU scalability. To minimize inter-server communication in Hydra, we observe that the subcarrier-parallel stage transforms the data dimension from antennas to users, and massive MIMO by nature uses much fewer users than antennas. Therefore, we delay inter-server communication until after the subcarrier-parallel stage, shuffling only a small fraction of the BBU's input data rate among its servers.

3. **Within one server.** Within a machine, Hydra affinitizes the processing of an OFDM subcarrier to a CPU core. This ensures that the same CPU cores process a subcarrier through multiple subcarrier-parallel BBU sub-stages, eliminating inter-core shuffling of intermediate outputs. Hydra also avoids centralized scheduling of BBU tasks, which reduces inter-synchronization overhead, and prevents a single thread from becoming a bottleneck. This allows Hydra to use up to 47% fewer CPU cores than Agora's design (Section 5.6).

Besides the above key ideas, we also dynamically increase or decrease the number of CPU cores used, to efficiently handle the varying demands in mobile networks and reduce the energy consumption. We build Hydra starting from Agora's open source implementation. Our evaluation with an RU emulator shows that the number of antennas and users that Hydra can handle scales with the number of servers. With three servers, Hydra handles 150×32 MIMO, which has 2.3x more antennas and 2x more users than the prior state of the art single-machine system (Agora). With a larger 18-node cluster of old servers, Hydra handles 256×32 MIMO. Our evaluation also shows Hydra reduces CPU use by 46% when the traffic demand is low, compared to the corresponding system without dynamic core scaling.

2 Background and motivation

Antennas at a multi-user massive MIMO RU receive wireless signals that are a combination of several users' transmissions. Each antenna has associated hardware that digitizes these signals into per-subcarrier IQ samples (typically represented as fixed-point complex values), assembles the IQ samples into packets, and transmits them to the BBU over a wired fronthaul link. The BBU's task is to recover the bits

transmitted by each user from these jumbled complex numbers.

Doing so requires a huge amount of signal processing on the IQ samples, including matrix operations and forward error correction. This high computation cost is justified by the corresponding improvements in spectral efficiency. Note that although server hardware is an important contributor to operating expenditure, spectrum is often the most valuable resource in mobile networks.

2.1 Massive MIMO basics

Traditional single-user base stations allows at most one user to communicate with the base station on a given frequency resource (i.e., a subcarrier), avoiding inter-user interference. Multi-user MIMO uses interference canceling to allow a mobile base station to serve multiple users concurrently on the same subcarrier. Multi-user MIMO exploits *spatial* diversity, which means that different users are separated in physical space and therefore have different channels to the RU. Massive MIMO refers to multi-user MIMO with a large number (typically 32 or more) of base station antennas.

An M×N massive MIMO configuration uses M RU antennas to simultaneously serve N user antennas. On a given subcarrier, we can represent the signals transmitted by the N user antennas using an N×1 complex-valued column vector $x_{N\times1}$. The signal $y_{M\times1}$ received by the RU is a mixture of all users' transmissions. y can be modeled as $\approx W_{M\times N} \times x_{N\times1}$, where W is the "channel matrix", i.e., $W_{i,j}$ is the wireless channel between antenna i and user j.

Zero-forcing receivers. The BBU's main task then is to jointly process the signals y from all RU antennas to recover the users' signals x. Importantly, the BBU can do this joint processing for each subcarrier in parallel. The joint processing consists of two steps. First, the BBU estimates the channel matrix W by using "pilot" transmissions from the users that have well-known numerical values, and are separated in time or frequency to avoid inter-user interference. Second, with the common "zero-forcing" approach, the BBU then computes the pseudo-inverse of the channel matrix W, as $H = (W^*W)^{-1}W^*$. For subsequent non-pilot data transmissions, the BBU recovers an approximation of x by computing $H \times y$, in a process called *equalization*.

After reconstructing x, the BBU performs *demodulation* to map the complex numbers to bits. The demodulated bits contain both user data bits and parity bits, appended by the radio protocol.

Finally, the BBU *decodes* the demodulated output via a forward-error correction (FEC) algorithm to produce the users' bits. Similar to Agora [17], Hydra uses 5G's Low Density Parity Check (LDPC) algorithm for decoding.

2.2 Massive MIMO baseband processing

Our goal in Hydra is to distribute massive MIMO BBU processing among a pool of servers using the fewest number

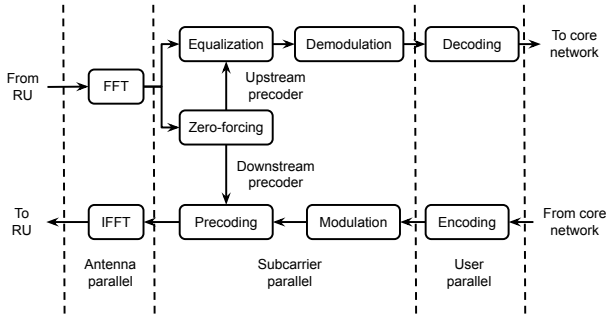


Figure 2: Massive MIMO processing pipeline.

of CPU cores and servers, achieved by minimizing distribution overheads from inter- and intra-server communication. We next discuss the two aspects of the massive MIMO processing pipeline that are crucial for designing a scalable distributed system: (1) the opportunity for distribution presented by the different types of parallelism in each stage, (2) and the scalability challenge posed by the need to shuffle data from one pipeline stage's output to the next stage's input (which we colloquially call the "data shuffling overhead").

Through the paper, we use the largest massive MIMO configuration supported by Hydra as a running **example** for exposition: 150×32 MIMO; with a typical 20 MHz configuration: 2048 subcarriers, out of which 1200 subcarriers carry data and the rest are used for guard bands; and 1 ms "slots" (discussed next).

Slots and symbols. Wireless protocols such as 4G and 5G divide time into slots. Each slot duration is typically further subdivided into 14 *symbol* durations. In each symbol duration, each RU antenna sends packets to the BBU containing IQ samples for all subcarriers. The radio protocol reserves pre-configured symbols for pilot signals from users, which are used for channel estimation. Similar to prior work [18, 28], we use the first symbol for pilots.

2.2.1 Types of parallelism

As noted by prior work [18, 28], massive MIMO baseband processing exhibits parallelism in different dimensions at different stages of the processing pipeline. This allows BBUs to divide the processing among multiple workers (i.e., CPU cores or servers). Figure 2 shows the three dimensions of parallelism for both uplink and downlink.

In the first antenna-parallel stage, the BBU performs FFT on the 150 antenna streams in parallel, converting time-domain IQ samples into frequency-domain samples. This step also eliminates the guard subcarriers and retains the 1200 subcarriers. The second frequency-parallel stage consists of three sub-stages: the BBU performs channel inversion, equalization, and demodulation for the 1200 subcarriers in parallel. The BBU may amortize the high cost of

matrix inversion by assuming that the channel matrix for some small configurable number of consecutive subcarriers is the same. In the third user-parallel stage, the BBU performs FEC decoding for each user independently.

2.2.2 Challenge: Inter-stage data shuffling

Massive MIMO processing is not perfectly parallelizable because the type of parallelism changes at each stage of massive MIMO processing, requiring shuffling the output of one stage to the input of another stage. One of our design goals in Hydra is to minimize this shuffling overhead.

Antenna-parallel to subcarrier-parallel shuffling. Consider a thread T_{fft} that has computed the FFT for a fronthaul packet in the antenna-parallel stage. At this point, T_{fft} has data for all 1200 subcarriers. T_{fft} must then transmit data from different subcarrier ranges to the subcarrier-parallel stage threads processing the corresponding subcarrier ranges. This transmission uses shared-memory for destination threads on the same machine, and over-the-network transmission for remote threads.

Subcarrier-parallel to user-parallel shuffling. Consider a thread T_{sc} that has finishes the subcarrier-parallel stage (i.e., demodulation) for its partition of subcarriers. At this point, T_{sc} has data for all users, and must transmit different user ranges to threads running user-parallel processing.

In Section 3, we discuss how Hydra avoids the overhead of both explicit and implicit *intra*-stage data shuffling in prior BBU designs.

2.3 The need for distributed computing

Our goal for building a scalable distributed design for massive MIMO BBUs is to provide a path for scaling to massive MIMO's ever-increasing computational demands. If massive MIMO vRANs are limited to a single server, they will suffer from limited mobile bandwidth, spectral efficiency, and have a less competitive feature set compared to traditional BBUs based on specialized hardware. Note that while this paper focuses on increasing antennas and users as the main driver for higher computation requirements, other important factors such as increasing the frequency bandwidth (e.g., 20 MHz to 100 MHz) and decreasing the slot size (e.g., 1 ms to 0.5 ms) also substantially increase the computation resources required and fronthaul traffic bandwidth.

PHY latency deadlines. The radio protocol's NACK (negative acknowledgment) turnaround time imposes a latency deadline on BBU processing. For example, in 4G and 5G, in case of an irrecoverable bit error on the uplink, the BBU must send a downlink NACK to the user within four slots (i.e., within 4 ms). In this work, we set Hydra's latency deadline to 2.5 ms at the 99.99-th percentile, to allow 1.5 ms for the MAC to schedule the downlink NACK.

High computational requirements. The number of CPU cores required to meet the BBU's latency deadline increases with MIMO dimensions, eventually exceeding the capacity of a single machine and necessitating a distributed design. For example, even after our optimizations, Hydra requires two servers to support 128×32 MIMO, and three servers to support 150×32 MIMO. Our 150×32 massive MIMO configuration requires 71 CPU cores. Note that although servers with very large numbers (100+) of high performance cores are available today, vRAN operators typically deploy smaller servers due to constraints such as power draw and fleet homogeneity. We explain these factors in detail next.

Limitation of single-machine systems. The CPU requirement of large MIMO configurations such as 150×32 (71 cores) is too high for a single vRAN server. This is because vRAN servers are deployed in small edge datacenters that have limited energy and space budgets, which precludes using beefy servers (e.g., quad-socket servers with 100+ cores). vRAN servers are typically single-socket or mid-range dual-socket servers. For example, HPE's servers targeted for vRAN have at most 28 CPU cores [14]. Similarly, Dell's reference architecture for vRAN has 40 cores per server [4].

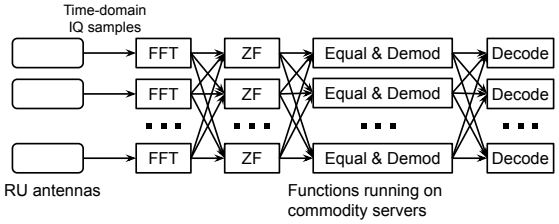
In addition, massive MIMO servers co-exist with vRAN servers handling other workloads, such as BBUs for non-massive RUs, and virtualized implementations of higher cellular protocol layers (e.g., MAC). Datacenter operators prefer maintaining a uniform fleet of servers, i.e., it is uncommon to deploy special high core-count servers for just one workload. Therefore, a distributed design that can support massive MIMO workloads in typical vRAN servers is useful.

Another advantage of not relying on high-end beefy servers, which we have currently not explored in this work, is cheaper fault tolerance. vRAN deployments must provide extremely high availability since they are part of the critical phone infrastructure. One way to limit vRAN downtime is to deploy some servers as hot backups. Maintaining a beefy backup server to guard against the failure of a single beefy server is more expensive compared to maintaining a smaller backup server to guard against failure of one of Hydra's servers.

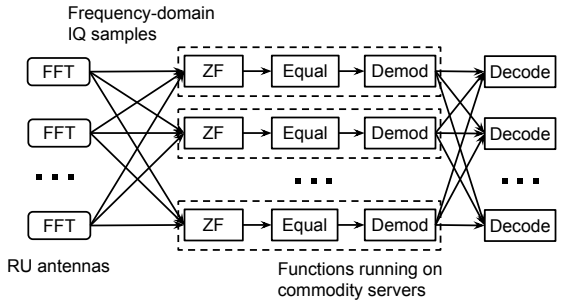
2.4 Limitations of prior distributed designs

BigStation [28] is the state-of-the-art design for virtualized distributed massive MIMO baseband processing. BigStation was designed around a decade ago for 4G MIMO processing, supporting up to 12 antennas and 12 users. BigStation's design (Figure 3) has two limitations:

High inter-core and inter-server communication. BigStation was designed for relatively small MIMO configurations, and aimed to meet latency deadlines with the weaker CPU cores available in 2012. Thus, BigStation aggressively distributes decomposable BBU tasks among CPU cores in the cluster (Section 3.2). As the MIMO dimension



(a) BigStation architecture



(b) Hydra architecture

Figure 3: Architecture comparison between BigStation and Hydra (uplink). Unlike BigStation, Hydra shuffles data only after the subcarrier-parallel stage. "ZF", "Equal", "Demod", and "Decode" are short for zero-forcing, equalization, demodulation, and LDPC decoding, respectively.

increases, the communication overhead in BigStation becomes significant and limits its scalability. In contrast, today we have higher MIMO dimensions and more powerful CPU cores that can individually complete the required MIMO operations within the radio protocol's deadline. Therefore, Hydra's design centers on keeping computation local on a CPU core to the extent possible.

Time-domain fronthaul traffic. BigStation was designed for older RUs that lacked FFT support (Section 3.1). To handle the large number of antennas in massive MIMO, BigStation's servers spend a correspondingly large amount of CPU cycles in (a) performing FFT and IFFT, and (2) sending, receiving, and shuffling IQ samples (Section 4). For example, in the uplink direction, the servers first receive time-domain IQ samples from antennas, perform FFT, and then shuffle the frequency-domain IQ samples among each other to enter the subcarrier-parallel processing stage.

2.5 Motivation and challenges for Hydra

The above limitations of single-server and distributed MIMO BBUs motivated us to create a new distributed design. The key challenge in Hydra is: how can we distribute massive MIMO BBU processing among a pool of workers (servers or CPU cores) with minimal overhead? An ideal design is to perfectly parallelize the workload among the BBU servers without any inter-server communication.

When splitting massive MIMO BBU processing for an RU with fronthaul traffic F_{bw} Gbps among N servers, the minimal network communication each server must handle is $C_{min} = F_{bw}/N$ Gbps. Our design achieves close to perfect parallelism with only 20% additional inter-server communication compared to C_{min} (Section 5.2).

3 Design

We next describe Hydra's three main design components. Our design focuses on reduces communication overhead, with three approaches: (1) using the ability of modern RUs to run FFT and split packets into subcarrier ranges (Section 3.1), (2) shuffling data between servers at a stage that minimizes inter-server traffic (Section 3.2), and (3) avoiding inter-core data movement and coordination within a server (Section 3.3).

For brevity, similar to prior work [18, 24, 29] we primarily focus on uplink processing, which is often more computationally intensive than downlink processing due to the higher cost of channel decoding compared to encoding. Interestingly, we find that downlink processing can be costlier than uplink on some server architectures (Section 5.2).

3.1 Scalable fronthaul traffic partitioning

Massive MIMO radios generate a high rate of fronthaul traffic. For a scalable design with minimal overhead, it is critical to not re-shuffle any substantial fraction of the fronthaul traffic between servers. Doing so adds overhead in terms of CPU cycles, latency, and datacenter network bandwidth use. We show that the new ability in modern RUs to run FFT and generate separate packets for different subcarrier ranges allows partitioning the fronthaul traffic among Hydra's servers with zero overhead. This reduces the amount of traffic that each server must handle before the subcarrier-parallel stage by up to 66.4% compared to BigStation.

Quantifying fronthaul bandwidth. For example, the fronthaul bandwidth in our running example configuration (Section 2.2)—150×32 MIMO, 20 MHz frequency bandwidth (2048 subcarriers, 1200 data subcarriers), 1 ms slots with 14 symbols—is 80.6 Gbps, assuming that the RU performs FFT and eliminates guard subcarriers: Each of the 150 antennas generates one packet with 1200 subcarriers (four bytes per IQ sample) for each of the 14 symbols in a slot. Therefore, the BBU receives $150 \times 1200 \times 4 \times 14$ bytes every millisecond, totaling 80.6 Gbps.

Note that most of the factors listed in Section 2.3 cause the fronthaul bandwidth to increase linearly. For example, using 100 MHz bandwidth with 0.5 ms slots is a common configuration in 5G deployments. This increases the fronthaul bandwidth by 5.5x to 444 Gbps by (1) increasing from 1200 to 3300 data subcarriers (4096 total subcarriers), and (2) doubling the rate at which the BBU receives packets.

To better describe the advantages of Hydra's fronthaul traffic partitioning approach, we begin by first discussing BigStation's approach. We compare the datacenter network

System	Receive rate	Transmit rate
BigStation	47.3 Gbps	12.9 Gbps
Hydra	20.2 Gbps	0 Gbps

Table 1: Comparison of datacenter network bandwidth used in BigStation and Hydra before the subcarrier-parallel stage. These numbers are for a 150-antenna RU and 20 MHz bandwidth with 1 ms slots.

bandwidth handled by each machine before the subcarrier-parallel stage in BigStation and Hydra, assuming each BBU design uses a cluster of four machines, and our running example MIMO configuration.

BigStation's approach BigStation uses RUs without FFT support, and therefore operates on time-domain IQ sample packets, with 2048 IQ samples each (one IQ sample per subcarrier). In our example MIMO configuration, the fronthaul bandwidth for BigStation is therefore $80.6\times2048/1200$ =137.5 Gbps. BigStation partitions the fronthaul traffic by antenna: each of the four BigStation servers receives packets for an exclusive range of 32 antennas, i.e., 137.5/4 = 34.4 Gbps.

BigStation's servers then run FFT on the time-domain IQ sample packets and drop the guard subcarriers, retaining 1200 subcarriers. To feed the next subcarrier-parallel stage (Section 2.2.2), each server must send 75% of the 1200 subcarriers to other machines. This corresponds to $32\times(0.75\times1200)\times4\times14$ bytes per millisecond, or 12.9 Gbps. Symmetrically, each server also receives 12.9 Gbps of shuffling input from the other three servers.

Hardware capabilities of modern radios The O-RAN alliance [9] defines specifications for various components and interfaces in 5G vRAN deployments. There are two hardware capabilities in modern O-RAN–compliant RUs that allow us to design new ways to partition fronthaul traffic. (1) FFT support. O-RAN's fronthaul specification [7] requires RUs to support FFT. Running FFT at the RU reduces fronthaul bandwidth requirement by dropping guard subcarriers at the RU. For example, in our 20 MHz configuration with 2048 subcarriers and 1200 data subcarriers, running FFT at the RU cuts down fronthaul bandwidth by 41%. Since FFT is cheap to implement, it is widely included in RUs. This O-RAN feature benefits massive and non-massive RUs alike (e.g., small cells in a city that have low-bandwidth connections to the BBU).

(2) O-RAN also requires the RU to support configurable "fragmentation" (Section 3.5 in the fronthaul spec [7]) of its IQ sample packets on both the uplink and the downlink. This is needed to support fronthaul networks with different MTUs (a fronthaul packet with 1200 subcarriers requires 4800 bytes, which exceeds Ethernet's typical 1500-byte MTU), and to allow the BBU to request only certain frequency ranges from the RU. The latter is useful for reducing fronthaul traffic during low load when only a few frequency resources are in use.

Hydra's approach We found a novel use case of these two RU abilities, which is different from what they were originally designed for (i.e., reducing fronthaul traffic or handling different MTUs): distributing massive MIMO fronthaul traffic among a pool of servers with zero overhead.

Fortunately, O-RAN RUs do not fragment packets at arbitrary boundaries, such as in the middle of an IQ sample. With such an implementation of fragmentation, Hydra's servers would need to re-shuffle some IQ samples between servers before the subcarrier-parallel processing stage. Instead, each fragment contains a contiguous range of subcarriers, and the BBU can configure these ranges over its control plane connection to the RU. In Hydra, we use as many equally-sized ranges as the number of servers. If the number of subcarriers is not a multiple of the number of servers, one server gets a slightly larger range than others.

For example, in a four-server Hydra cluster, we configure the RU to send four packets per antenna in each symbol duration. The four packets contain IQ samples for data subcarrier ranges 0–299, 300–599, 600–899, 900–1199. We then route the i^{th} packet to server i. Each Hydra server therefore receives 80.6/4 =20.2 Gbps and sends no datacenter network traffic before entering the subcarrier-parallel stage.

Table 1 compares the amount of traffic received and sent by each server before the subcarrier-parallel stage in BigStation and Hydra for our example configuration. Hydra's total bi-directional bandwidth requirement per-server (20.2 Gbps) is 66.4% percent lower than BigStation's (60.2 Gbps). Our evaluation shows that using FFT-capable RUs with subcarrier range fragmentation reduces the number of CPU cores needed by Hydra by up to 46% (Section 5.5).

3.2 Scalable PHY computation partitioning

After partitioning subcarriers, Hydra still needs to run the subcarrier-parallel stage (i.e., zero-forcing, equalization, and demodulation), and the user-parallel stage (i.e., decoding). There are several possible approaches to partitioning this remaining computation among Hydra's servers. We observe that the massive MIMO processing pipeline progressively reduces the amount of data transferred between pipeline stages. Hydra minimizes the amount of inter-server data shuffling by delaying shuffling to the last stage of the MIMO processing pipeline (decoding). Compared to BigStation's design, Hydra's servers shuffle up to 42% less data (Section 3.2.2).

3.2.1 Hydra's approach

Recall that in our running example, the i^{th} Hydra server S_i receives IQ samples for subcarriers $[300\times i, 300\times(i+1))$. S_i runs all the subcarrier-parallel processing sub-stages for these 300 subcarriers locally, i.e., without shipping any com-

putation to remote servers. After demodulation, S_i creates 32 per-user output buffers, each with 300 entries for the corresponding users' transmissions on each subcarrier. The decoding for 75% of these users happens on other servers, so S_i ships its outputs for those users over the network.

Rationale. The transition point between subcarrier-parallel and user-parallel stages offers an efficient point for shipping computation across servers. This is because the subcarrier-parallel stage reduces the amount of data flowing through the BBU's uplink pipeline: The equalization step for a subcarrier transforms per-antenna samples into per-user samples, and the number of antennas in massive MIMO is substantially larger than the number of users. For example, 64×8 is the typical massive MIMO configuration in today's 5G deployments. In our running 128×32 massive MIMO example, equalization shrinks the pipeline's flow by 4x.

3.2.2 BigStation's approach

The main alternative to our approach is computation and data shipping even within the subcarrier-parallel stage. For example, unlike Hydra, the computation for a given subcarrier in BigStation happens on different servers: BigStation reserves a set of its servers for only matrix inversion. These servers then ship the computed inverses to other servers running equalization and demodulation.

Comparison with Hydra. In our running example with 150×32 MIMO, we assume that groups of 32 subcarriers (as many as the number of users to avoid inter-user interference during pilot transmission) have the same channel matrix. Therefore, there are 1200/32 channel matrices, and each matrix is $150 \times 32 \times 8 = 38400$ bytes in size. Shipping these matrices over the network in each millisecond slot duration requires 11.5 Gbps.

Shuffling data between the subcarrier-parallel stage and the user-parallel stage for this MIMO configuration cumulatively requires 16 Gbps for a three-server BBU (Section 5.2). This shuffle is required in both Hydra and BigStation, but it is the only shuffle required in Hydra. Therefore, Hydra's inter-server shuffle bandwidth (16 Gbps) is 42% lower than BigStation's (11.5 Gbps + 16 Gbps) for this configuration.

Alternative approaches. Similarly, early versions of our system aimed to maximize parallelism in the MIMO BBU by dynamically shipping individual matrix inversion and multiply operations over the cluster. Our thinking was in line with BigStation's hypotheses [28, Section 5.3] that for very large MIMO systems, the matrix inverse and multiply operations may need to be partitioned across servers. However, such an approach is unnecessary and inefficient: distributing individual MIMO matrix operations is unnecessary because modern CPU cores are individually powerful enough to meet the BBU's deadline. For example, computing the pseudo-inverse of a 150×32 channel matrix takes only around 150 µs on our servers. Our evaluation shows that confining a subcarrier's processing to a single server (a single core) works well on today's hardware. In addition, shipping the matrix operations adds overhead by shipping a huge amount of matrix contents over the network, requiring similar bandwidth to the fronthaul bandwidth.

3.3 Scaling within a machine

In our goal to build a scalable distributed system for massive MIMO baseband processing, we also had to create new optimizations for the processing within a single machine. We found that our baseline Agora system has a large amount of overhead from inter-core data movement and synchronization. We next describe two optimizations to reduce inter-core data movement and synchronization that we made on top of Agora. Our evaluation shows that these optimizations are crucial: without them, for some large MIMO configurations, Hydra either fails to support the configuration, or requires over 2x more CPU cores (Section 5.6).

Subcarrier-to-core affinity Recognizing the high cost of inter-core communication, we design Hydra to use the same CPU core for all the subcarrier-parallel sub-stages for a given subcarrier. In contrast, Agora centers its design around fine-grained task distribution, so a random core runs any individual matrix inverse or matrix multiply. Although this is a straightforward way of parallelizing MIMO processing that provides flexibility in allocating tasks to cores, it incurs a large amount of inter-core communication.

In Agora, the CPU core that computes the channel matrix inverse for a subcarrier (C_i) is almost always different from the CPU cores that run equalization and demodulation for that subcarrier (C_e). Note that a given channel matrix inverse computed from the pilot symbols is used for equalization in 13 subsequent data symbols. This creates overhead by repeatedly moving the computed matrix inverse from C_i's private caches to C_e's caches. It also reduces the cache efficiency of all cores, since the same matrix contents are duplicated in several caches.

In Hydra, the same CPU core performs channel matrix inversion, equalization, and demodulation for a given subcarrier. This eliminates inter-core cache movement and duplication of cached data.

No central coordinator thread. Agora uses a coordinator-worker thread design, in which a single coordinator thread communicates with worker threads via shared-memory queues. The coordinator thread queues task descriptors to the workers (e.g., the address and dimension of a source matrix to invert), and receives completions from workers. We find that in Agora's design, the worker threads spend a substantial amount of time blocked and waiting for work from the coordinator. This happens because the coordinator must schedule a large number of tasks, restricting performance. In contrast, Hydra's threads use shared-memory counters to track dependencies in the

Task	Without HT	With HT
Invert a 150×32 matrix	2.9 K/sec	4.8 K/sec
Equalization (32×150 times 150×1)	0.96 M/sec	1.23 M/sec
Precoding (150×32 times 32×1)	0.67 M/sec	0.72 M/sec
LDPC decoding (one UE)	20.2 K/sec	23.3 K/sec
LDPC encoding (one UE)	48.5 K/sec	60.0 K/sec

Table 2: Comparison of single-core processing rate with and without Hyper Threading (HT) for 150×32 MIMO.

MIMO pipeline, avoiding the coordinator bottleneck. In addition, since we also use subcarrier-to-core affinity, there are over 10x fewer cross-core task dependencies in Hydra than in Agora.

3.4 Downlink processing

The BBU's downlink pipeline is the reverse of the uplink pipeline. A separate MAC layer sends per-user data to Hydra's user-parallel threads, which perform LDPC encoding, and send the corresponding subcarrier ranges to Hydra's subcarrier-parallel workers. Subcarrier-parallel workers then apply the precoder matrix that they computed during uplink processing (the precoder is the transpose of the channel matrix inverse) to transform per-user streams to per-antenna streams. The packet I/O threads on each machine then combine the outputs of each subcarrier-parallel worker to generate one packet for the machine's assigned subcarrier range, which is then sent to the RU.

4 Implementation

We implement Hydra in C++ for Linux, building on top of Agora's open-source codebase. Similar to Agora, (1) we use Intel MKL for matrix operations accelerated with AVX-512 SIMD instructions, (2) we use Intel's FlexRAN library [6] for LDPC decoding and encoding.

Thread types in Hydra. A Hydra deployment consists of one or more Hydra processes running on different servers in a cluster. Each Hydra process launches three sets of threads, each pinned to a different core: (1) packet IO threads, (2) subcarrier-parallel threads, and (3) user-parallel threads. Packet IO threads send and receive fronthaul traffic, and shuffle BBU pipeline data between servers. Each subcarrier-parallel and user-parallel thread is assigned a static range of subcarriers or range of users, respectively. For each MIMO dimension, we use the fewest number of threads for each thread type (currently determined manually) required to handle the maximum workload.

Hyper Threading. While Agora disables Hyper Threading (HT), we find that enabling it improves the performance of massive MIMO processing by reducing the negative impact of memory stalls. Massive MIMO processing generates a large memory footprint (e.g., 1200 150×32 matrices, or 11.5 MB), causing misses in the CPU's L1–L3 caches, which reduce CPU efficiency. Hyper Threading hides the impact of these memory stalls by overlapping memory accesses with compute, e.g., by allowing one logical thread to use a SIMD unit while another logical thread is stalled. Table 2 shows that for 150×32 MIMO, using HT improves a single core's throughput by 7.5%–65.5% for different PHY routines measured in isolation in micro benchmarks. For end-to-end runs, we were able to fit 150×32 MIMO processing in three servers only with HT enabled. For smaller MIMO configurations that we were able to test both with and without HT, using HT reduces the number of physical CPU cores needed by Hydra, e.g., from 68 to 53 cores for uplink processing for 128×32 MIMO.

Dynamic CPU core utilization. Since mobile networks experience highly variable workloads, it is important for the BBU to scale its energy consumption with the workload [1]. For example, cell sites in residential areas have high utilization during the day time, but almost no utilization at night. Hydra scales its CPU usage with the workload as follows.

For every slot, the MAC layer (not included in our system yet) communicates the set of users active on each subcarrier to Hydra. If during a slot with low load, the base station has fewer active users than the number of users permitted by the MIMO dimension, Hydra puts the corresponding user-parallel threads to sleep. Similarly, if a slot's MAC configuration has some subcarriers not assigned to any user, Hydra puts the corresponding subcarrier-parallel threads to sleep. While this approach is fairly simple, we believe that it works well because mobile networks experience highly bursty workload patterns, with significant periods of zero load. For example, recent measurements show that a 4G cell is fully idle in 75% of the slots [20]. During zero-load periods, Hydra disables most of its threads, keeping only the packet I/O threads active. The CPU cores yielded by Hydra may be used by other co-located latency-tolerant edge workloads such as machine learning and analytics [20].

5 Evaluation

This section presents our evaluation of Hydra's performance, the effectiveness of our design choices, and comparisons with design choices made by prior massive MIMO baseband processing systems (i.e., Agora and BigStation).

For evaluation, we created a complete version of BigStation based on the original design [29]. To focus the evaluation on the distributed system design differences between BigStation and Hydra, we also implement all of Hydra's single-machine optimizations for BigStation.

5.1 Evaluation setup

5.1.1 Server setup

We run our evaluation in two clusters. For most experiments, we use a "main" cluster of four commodity servers, with three servers running our distributed BBU, and one server acting as a fronthaul traffic generator emulating an

RU (Section 5.1.2). All servers are connected to an Arista 7060 switch with 100 GbE single-port Mellanox ConnectX-5 NICs. Each server has two Intel Xeon Silver 4216 CPUs (2.1 GHz, AVX512 support), with 16 cores per CPU. We use at most 29 cores per server to leave some cores for the OS to avoid kernel thread starvation.

To demonstrate our design's scalability to more servers than available in our main cluster, we use another cluster in CloudLab [19] consisting of 27 servers, with up to 18 servers for the BBU, and nine servers for emulating the RU. These servers are less powerful than our main cluster's servers. All servers are connected to Mellanox 2410 switches with a Mellanox ConnectX-4 25 GbE NIC. Each server has one ten-core Intel E5-2640v4 CPU (2.4 GHz, no AVX512 support).

As is typical in vRAN systems [3, 18], we configure each server to reduce jitter: we run our processes as real-time processes with the highest scheduling priority, and remap OS interrupts to an unused core. Our experiments run in a dedicated cluster without network congestion and therefore experience no packet loss. Unless specified otherwise, the experiments run in the main cluster.

5.1.2 Emulated fronthaul traffic generator

Since O-RAN-compatible massive MIMO radios are not readily available today, we emulate the fronthaul traffic with a software-based generator, using Agora's DPDK-based generator as a starting point. The generator applies a Rayleigh fading channel with Gaussian noise (25 dB signal-to-noise ratio). Agora's generator emulates a basic RU without FFT support, transmitting time-domain IQ samples; we modify it to emulate an O-RAN radio: our generator runs FFT, discards guard subcarriers, and splits packets into multiple subcarrier ranges, one per Hydra server. In addition, for the packet for a given subcarrier range, the generator uses the network address (IP and MAC address) for the corresponding Hydra server.

In the CloudLab cluster, which has 25 GbE, the fronthaul bandwidth exceeds a single server's NIC bandwidth for MIMO configurations with over 46 antennas. To overcome this, we split the traffic generation for the antennas across multiple servers. All servers are time-synchronized to a sub-microsecond accuracy with PTP, and agree on the first slot's start time during initialization. We also add generator support to change the set of active subcarriers and users to emulate high and low load scenarios.

5.1.3 Wireless parameters

Our wireless settings are similar to Agora: all experiments use a 20 MHz configuration with 1200 data subcarriers (2048 total subcarriers), 1 ms slot duration, and 64 QAM modulation. We use 1/3 LDPC code rate and base graph #1, with a LDPC lifting size ("Z") up to 104. This configuration results in 29.7 Mbps data rate per user, or 950 Mbps for 32 users. Since our primary focus is performance, our experiments use the peak load where all subcarriers and users are active, unless mentioned otherwise.

5.2 End-to-end performance

Figure 4 shows the number of CPU cores and servers needed by Hydra and BigStation to support different massive MIMO configurations. We show the numbers for both uplink and downlink processing. We run the experiment for 100 seconds, spanning 100k 1 ms slots. To support a MIMO dimension, the BBU must satisfy two constraints: (1) the BBU's 99.99th percentile latency must be below 2.5 ms (Section 2.3), and the BBU must have a throughput of one slot per slot duration (1 ms in our case) to keep up with the RU. Hydra supports up to 150×32 MIMO with three BBU servers. For 150×32 MIMO, Hydra uses 71 cores for uplink processing, or 83 cores for downlink processing.

Interestingly, we find that in our main cluster, Hydra's downlink processing is more expensive than uplink. This is the opposite of measurements in the Agora paper, as well as our CloudLab measurements for Hydra (Section 5.4.3). This happens because downlink precoding (0.72 M/s per core for 32×150 by 150×1 multiplications) is more expensive than uplink equalization (1.23 M/s per core for 150×32 by 32×1 multiplications) on our main cluster (Table 2).[1] In our CloudLab cluster, the lack of AVX512 instructions reverses this effect (i.e., downlink becomes cheaper than uplink) by making LDPC decoding far more expensive than LDPC encoding: decoding is 10x more expensive than encoding on CloudLab, compared to 2.5x more expensive on our main cluster.

5.2.1 Comparison with BigStation

BigStation supports only up to 128×16 MIMO with the three servers. For MIMO configurations that BigStation supports, Hydra uses only around half the CPU cores for uplink processing, and between 30–40% fewer CPU cores for downlink processing. BigStation's worse performance comes from two factors. First, BigStation spends additional CPU cycles for running FFT in software instead of the RU, and shuffling a larger amount of data between servers than Hydra. Second, the higher network I/O and data shuffling generates more memory pressure and inter-core communication than Hydra, reducing BigStation's compute efficiency.

We provide a detailed accounting of Hydra's and BigStation's CPU usage below, with 128×16 downlink processing (the most challenging downlink configuration supported by BigStation) as the example.

- **Packet I/O.** BigStation uses 24 cores for packet I/O, compared to only four for Hydra.
- **IFFT.** BigStation uses six cores for IFFT processing, whereas Hydra uses the RU's ability to perform IFFT.

[1] We are investigating the root cause of this difference by studying Intel MKL's implementation of matrix multiplication.

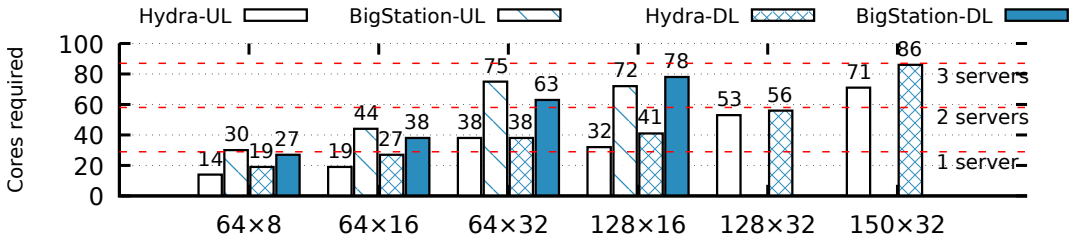


Figure 4: Number of cores and servers required to support different massive MIMO settings for Hydra and BigStation in uplink (UL) and downlink (DL) mode. BigStation supports up to 128×16, so the bars for larger configurations are not shown.

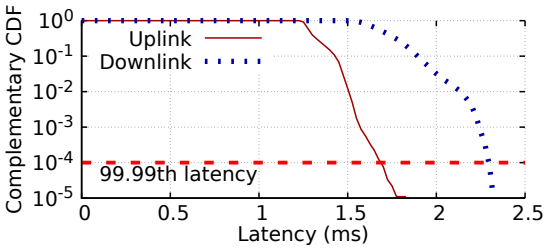


Figure 5: Complementary CDF of Hydra's latency for 150×32 uplink and downlink processing.

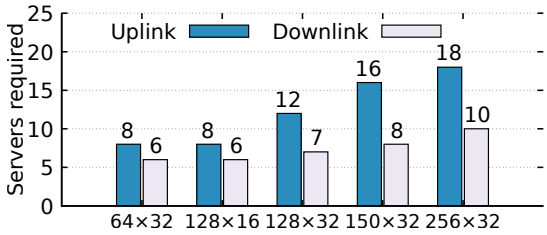


Figure 6: Number of servers required to support different massive MIMO settings in the CloudLab cluster.

- **Subcarrier processing.** BigStation uses 33 cores for subcarrier-parallel processing (six cores for zero-forcing, and 27 cores for precoding). Hydra uses 31 cores for its combined subcarrier processing stage.
- **LDPC encoding.** BigStation uses nine cores for LDPC encoding, compared to six for Hydra.

5.3 Comparison with Agora

Hydra supports Agora's largest 64×16 MIMO configuration with one server for both the uplink and downlink. Different from Agora's single-server design, Hydra allows using two servers to support 128×32 MIMO, and three servers to support 150×32 MIMO.

For a single-server performance comparison, we compare Hydra's performance with the numbers published in the Agora paper [18]. This is because we were unable to reproduce numbers comparable to those reported in Agora due to hardware differences, e.g., we use weaker CPUs (16-core Xeon Silver 4216, $900 per CPU) than those used in Agora's evaluation (16-core Xeon Gold 6130, $1900 per CPU). For 64×16 uplink processing, Hydra uses 19 CPU cores compared to Agora's 28 (including Agora's two packet I/O cores); for downlink processing, Hydra uses 27 cores compared to Agora's 23.

5.4 Hydra's performance details

5.4.1 Tail latency

Figure 5 shows that Hydra successfully meets our latency target of sub-2.5 ms 99.99th percentile latency for Hydra's largest-supported 150×32 MIMO configuration. For the uplink, Hydra's *maximum* latency is only 1.8 ms, and its 99.99-th percentile tail latency is 1.7 ms. For the downlink, Hydra's maximum and 99.99% latency are both 2.3 ms.

5.4.2 Additional network traffic

For 150×32 MIMO, Hydra processes 80.6 Gbps of fronthaul traffic, and cumulatively shuffles only 16 Gbps among servers, or 20% additional traffic. (Since there are three servers, each server transmits two-thirds of 32×400×6 bytes per data symbol. Since we use 64 QAM modulation, our demodulation stage represents each subcarrier's sample with 6 bits.)

5.4.3 Server scalability

We use the CloudLab cluster to study how Hydra's design scales with an increasing number of servers. Figure 6 shows how Hydra supports higher massive MIMO dimensions as the number of servers increases in the CloudLab cluster. Hydra supports 256×32 MIMO with 18 servers for uplink processing, or with 10 servers for downlink processing. For the regime studied, the number of servers needed for uplink processing scales roughly linearly with the number of antennas: for 32 users, Hydra needs 8, 12, and 18 servers for 64, 128, and 256 antennas, respectively. There is room for further scaling since Hydra does not hit a scalability bottleneck at 256×32; this scale was limited by only the number of CloudLab servers we managed to reserve.

Different from our main cluster, downlink processing is cheaper than uplink processing in the CloudLab cluster. This is primarily because LDPC decoding is 10x more ex-

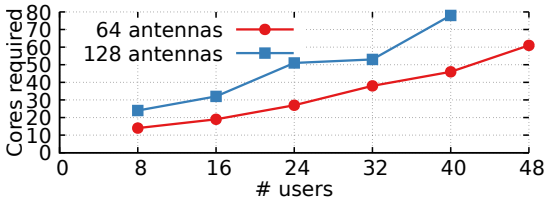


Figure 7: Minimum number of CPU cores required to support different massive MIMO settings.

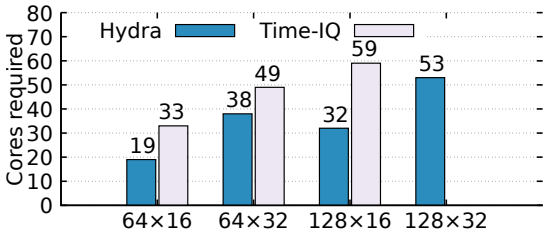


Figure 8: CPU cores needed by Hydra and Time-IQ for different massive MIMO configurations (uplink).

pensive than encoding on CloudLab servers, but only 2.5x more expensive on our main cluster.

5.4.4 User scalability

Figure 7 shows the minimum of CPU cores required to support an increasing number of users for two antenna configurations: 64 antennas and 128 antennas. We find that Hydra can scalably support more users by using more cores. Using LDPC accelerators (e.g., Intel's ACC100 accelerators) can allow Hydra to support even more users.

5.5 Benefits of leveraging RU features

To quantify the performance benefits of offloading FFT and subcarrier range splitting to the RU, we created a variant of Hydra called "Time-IQ" that works with time-domain IQ samples. Time-IQ runs FFT in software on the BBU servers to generate frequency-domain IQ samples, which it then shuffles among the servers for the subcarrier-parallel stage. Figure 8 compares the number of cores needed by Hydra and Time-IQ to support four different massive MIMO settings. Hydra uses 42%, 22%, and 46% fewer CPU cores for the 64×16, 64×32, and 128×16 configurations, respectively. Time-IQ is unable to support the 128×32 MIMO configuration with the 87 cores available in our cluster, whereas Hydra supports this configuration with 53 cores.

Next, we then run both Time-IQ and Hydra for 128×16 massive MIMO using 59 cores (the minimum CPU cores required by Time-IQ) and measure the 99.99-th tail latency breakdown. Figure 9 shows the 99.99-th percentile completion time for each of the three pipeline stages (the antenna-parallel FFT stage, the subcarrier-parallel stage, and the user-parallel decoding stage). Time-IQ has higher latency than Hydra due to the additional FFT processing in software, and a longer subcarrier-parallel stage. Time-IQ's subcarrier-parallel stage is longer because it must shuffle frequency-domain IQ samples between the antenna-parallel stage and the subcarrier-parallel stage.

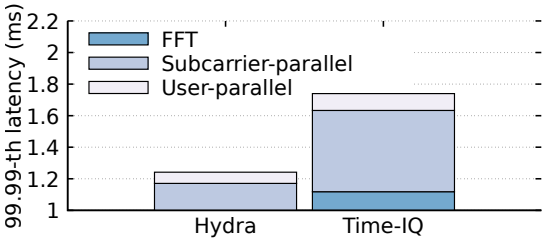


Figure 9: 99.99-th tail latency breakdown for Hydra and Time-IQ design for 128×16 MIMO (uplink) with 59 cores. The figure starts at 1 ms because it takes 1 ms to receive all IQ samples from RU antennas.

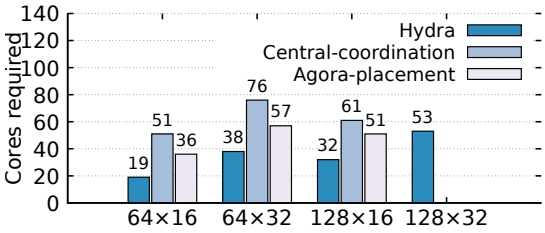


Figure 10: CPU cores needed by Hydra, Central-Coordination, and Agora-Placement for different massive MIMO dimensions (uplink).

5.6 Impact of intra-server optimizations

We next evaluate the effectiveness of our optimization to reduce inter-core communication (Section 3.3): affinitizing the processing of subcarriers to CPU cores, and avoiding a central coordinator thread for task scheduling. We created two variants of Hydra for this measurement: The first variant, called "Agora-Placement," works without a coordinator thread, but uses Agora's random assignment of tasks to CPU cores, which increase inter-core data movement and reduces cache effectiveness. The second variant, called "Central-Coordination," affinitizes subcarrier processing to CPU cores, but uses a coordinator thread to schedule tasks to workers. Figure 10 shows that reducing inter-core communication and avoiding centralization of task coordination logic is crucial for performance. In addition, the two variants are unable to support 128×32 MIMO with three servers.

For example, using a coordinator thread for task scheduling can more than double the number of CPU cores needed. We verify that this happens because of the large amount of time that worker threads in Central-Coordination spend in waiting for work from the coordinator thread. For example, with 128×32 MIMO and 53 cores (the minimum needed by Hydra to support 128×32), workers cores in the Central-

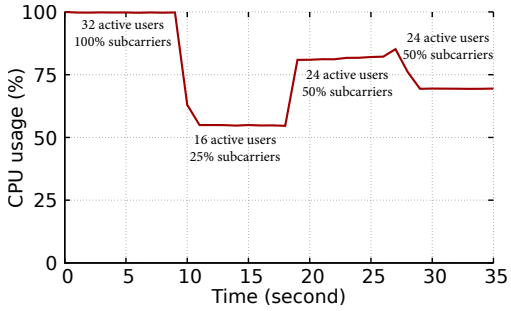


Figure 11: Hydra's CPU usage with a dynamic workload.

Coordination design spend around 0.5 ms on average in every millisecond slot waiting for the coordinator. In contrast, Hydra's workers spend only 0.14 ms on average waiting on shared-memory counters to saturate.

5.7 Dynamic CPU core scaling

We next evaluate the efficiency of Hydra's dynamic CPU core scaling mechanism. We run Hydra under 150×32 massive MIMO with 1200 data subcarriers, and dynamically change the number of active users and active data subcarriers. We change the workload in four stages, and each stage lasts for 8–10 seconds. The four stages are 1) 32 active users and 100% active subcarriers, 2) 16 active users and 25% active subcarriers, 3) 24 active users and 50% active subcarriers, and 4) 24 active users and 25% active subcarriers.

In a production RAN, the MAC layer sends commands to the PHY informing it about the number of active users and subcarriers in every slot; the PHY can use this information to adjust its CPU utilization [20]. Since we do not currently have a MAC layer, Hydra servers read per-slot configuration from a configuration file. Figure 11 shows the real-time CPU usage of Hydra over time, normalized to 71 cores at 100% load. Hydra first utilizes full CPU resource for the first stage, and reduces the CPU usage to 54% in the second stage. Hydra then dynamically changes the CPU usage to 80% and 70% in the third and the fourth stage.

6 Related Work

Software-based RAN processing. The use of commodity servers for high-performance PHY processing was pioneered by Sora [27], which demonstrated the use of modern CPU features such as SIMD for wireless signal processing for WiFi. The Sora project later led to BigStation [28], which was the first to use a distributed system to handle the high computation requirements of multiuser MIMO. Agora is a more recent project that focuses on massive MIMO processing within a single server. Hydra builds upon these designs by combining the single-machine design of Agora with ideas from BigStation, but focuses on minimizing the overheads in distributing massive MIMO computation. Intel's FlexRAN [3] is a production-grade single-machine PHY implementation is 5G NR-compliant and is used in large-scale vRAN deployments [10]. However, FlexRAN is closed-source, with a few open-source components like their LDPC encoder and decoder. Hydra's design could benefit from FlexRAN's other high-performance signal processing blocks, such as matrix inversion and demodulation.

Hardware-based RAN processing. The LuMaMi testbed [24] is a massive MIMO processing system that uses specialized hardware (e.g., FPGAs and PCIe switches). LuMaMi can handle 100×10 massive MIMO with 0.5 ms slots. While LuMaMi and Hydra cannot be compared apples-to-apples, it is interesting to note that Hydra can handle a substantially larger MIMO configuration (i.e., 150×32), although with a more relaxed latency deadline (1 ms slots). We believe that comparing software-only and hardware-based approaches for massive MIMO processing is an interesting avenue for future research.

Quantum computing approaches such as QuAMax [23] and ParaMax [22] have recently been proposed to tackle the high computational cost of massive MIMO. While our work uses linear MIMO methods (i.e., zero-forcing equalization and precoding), quantum-based approaches can handle more expensive non-linear methods like sphere decoding [21, 25]. Since sphere decoding can be too expensive for a single server, Hydra's techniques may be used to distribute the work among multiple servers.

7 Conclusion

We have presented the design of Hydra, a new distributed design for scalable massive MIMO processing in software. Hydra focuses its design on reducing the overhead in distributing massive MIMO computation among a pool of servers. Our design leverages features of modern RUs in novel ways to partition the fronthaul traffic with zero overhead, uses an efficient split for shuffling inter-server data between the MIMO pipeline's stages, and reduces inter-core communication and coordination for processing within a machine. The result is that Hydra can support much larger MIMO configurations than prior state-of-the-art, demonstrating support for 150×32 MIMO for the first time in software. Importantly, we have demonstrated that massive MIMO processing can be efficiently distributed over multiple servers, using only 20% additional network I/O compared to the required fronthaul traffic. We believe that our design can be used to scalably support even more challenging MIMO configurations in the future.

Acknowledgments. We thank the NSDI reviewers for their helpful feedback. We are grateful to Jian Ding and Rahman Doost-Mohammady for their feedback, and help with the Agora code. We also thank Lin Zhong for early discussions on the project. Junzhi Gong and Minlan Yu are supported in part by the NSF CNS-1955422 and CNS-1955487.

References

[1] A technical look at 5G energy consumption and performance. https://www.ericsson.com/en/blog/2019/9/energy-consumption-5g-nr.

[2] AirSpan 7200: Massive Throughput in a Single, Compact Unit OpenRANGE 7200. airspan.com/5g-products/.

[3] An Overview of FlexRAN* Software Wireless Access Solutions. https://software.intel.com/content/www/us/en/develop/videos/an-overview-of-flexran-sw-wireless-access-solutions.html.

[4] Building an open vRAN Ecosystem. https://www.delltechnologies.com/asset/en-us/solutions/service-provider-solutions/technical-support/altiostar-redhat-nec-and-dell-technologies-vran-solution-reference-architecture.pdf.

[5] Dish selects Fujitsu, Altiostar for 5G radios, Open vRAN. https://www.fiercewireless.com/operators/dish-selects-fujitsu-altiostar-for-5g-radios-open-vran.

[6] FlexRAN LTE and 5G NR FEC Software Development Kit Modules. https://software.intel.com/content/www/us/en/develop/articles/flexran-lte-and-5g-nr-fec-software-development-kit-modules.html.

[7] O-RAN Fronthaul Control, User and Synchronization Plane Specification v6.0. https://www.o-ran.org/specification-access.

[8] Open RAN and the mission to crack massive MIMO. https://www.lightreading.com/open-ran/open-ran-and-mission-to-crack-massive-mimo/d/d-id/768081.

[9] Operator Defined Open and Intelligent Radio Access Networks. https://www.o-ran.org/.

[10] Rakuten Mobile and NEC to Build Open vRAN Architecture in Japan. https://global.rakuten.com/corp/news/press/2019/0605_01.html.

[11] Telefonica invests in vRAN vendor Altiostar. https://www.fiercewireless.com/tech/telefonica-invests-vran-vendor-altiostar.

[12] T-Mobile Achieves Mind-Blowing 5G Speeds with MU-MIMO. https://www.t-mobile.com/news/network/t-mobile-achieves-mind-blowing-5g-speeds-with-mu-mimo.

[13] Vodafone starts trials of OpenRAN in Europe and Africa. https://www.gsma.com/futurenetworks/digest/vodafone-starts-trials-of-openran-in-europe-and-africa/.

[14] vRAN 2.0 on HPE Infrastructure. https://h50146.www5.hpe.com/products/servers/document/pdf/edgeline/vran2.0.pdf.

[15] Open RAN Alliance. O-RAN: towards an open and smart RAN. *white paper, October*, 2018.

[16] Robin Chataut and R. Akl. Massive mimo systems for 5g and beyond networks—overview, recent trends, challenges, and future research direction. *Sensors (Basel, Switzerland)*, 20, 2020.

[17] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. Agora: Real-time massive MIMO baseband processing in software. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 232–244, 2020.

[18] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. Agora: Real-time massive MIMO baseband processing in software. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 232–244, 2020.

[19] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[20] Xenofon Foukas and Bozidar Radunovic. Concordia: teaching the 5G vRAN to share compute. In Fernando A. Kuipers and Matthew C. Caesar, editors, *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, pages 580–596. ACM, 2021.

[21] Chin-yun Hung and Tzu-hsien Sang. A sphere decoding algorithm for mimo channels. In *2006 IEEE International Symposium on Signal Processing and Information Technology*, pages 502–506, 2006.

[22] Minsung Kim, Salvatore Mandrà, Davide Venturelli, and Kyle Jamieson. Physics-inspired heuristics for soft mimo detection in 5g new radio and beyond. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 42–55, New York, NY, USA, 2021. Association for Computing Machinery.

[23] Minsung Kim, Davide Venturelli, and Kyle Jamieson. Leveraging quantum annealing for large mimo processing in centralized radio access networks. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 241–255, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Steffen Malkowsky, João Vieira, Liang Liu, Paul Harris, Karl Nieman, Nikhil Kundargi, Ian C. Wong, Fredrik Tufvesson, Viktor Öwall, and Ove Edfors. The world's first real-time testbed for massive mimo: Design, implementation, and validation. *IEEE Access*, 5:9073–9088, 2017.

[25] Konstantinos Nikitopoulos, Juan Zhou, Ben Congdon, and Kyle Jamieson. Geosphere: Consistently turning mimo capacity into throughput. *SIGCOMM Comput. Commun. Rev.*, 44(4):631–642, aug 2014.

[26] Clayton Shepard, Jian Ding, Ryan E Guerra, and Lin Zhong. Understanding real many-antenna MU-MIMO channels. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 461–467. IEEE, 2016.

[27] Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M Voelker. Sora: high-performance software radio using general-purpose multi-core processors. *Communications of the ACM*, 54(1):99–107, 2011.

[28] Qing Yang, Xiaoxiao Li, Hongyi Yao, Ji Fang, Kun Tan, Wenjun Hu, Jiansong Zhang, and Yongguang Zhang. BigStation: Enabling scalable real-time signal processing in large MU-MIMO systems. volume 43, pages 399–410. ACM New York, NY, USA, 2013.

[29] Qing Yang, Xiaoxiao Li, Hongyi Yao, Ji Fang, Kun Tan, Wenjun Hu, Jiansong Zhang, and Yongguang Zhang. BigStation: Enabling scalable real-time signal processingin large MU-MIMO systems. *ACM SIGCOMM Computer Communication Review*, 43(4):399–410, 2013.