Research Statement

Michael J. Freedman

I create distributed systems that handle the large scale, dynamism, and critical importance of our emerging computing infrastructure. Rather than primarily focusing on system performance, my research typically involves (i) the introduction of new system functionality; (ii) the design of algorithms, protocols, and data structures that rigorously provide robustness, security, and assurance; (iii) the art of system design for cleaner abstractions and easier management; and (iv) the development of working prototypes and even production deployments that have shaped industry and reached millions of users.

With the growth of Internet content and the rise of online "cloud-hosted" services, the past fifteen years has witnessed a profound change in the way people use computers and the Internet. Users don't manage their own data or email; this happens automatically with storage spread across the "cloud." Popular content isn't served from a single webserver, but rather from worldwide networks of coordinating content delivery servers. New enterprises don't even need to buy physical machines to deploy new technologies, they can rent resources from public cloud providers. But rather than the 1960's "utility computing" vision of services running on a single, big-iron supercomputer, scalable services are being realized within datacenters consisting of tens to hundreds of thousands of unreliable, commodity machines. For performance and fault tolerance, services may stretch across many such sites worldwide. Software and distributed protocols help these complex systems scale and provide their desired properties.

Unfortunately, in the race to adopt new economic models and build ever larger systems, hard technical problems have remained unanswered, or engineers thought certain tradeoffs—about consistency, security and privacy, assurance, and management—were fundamental. My research has challenged previously held beliefs about how to build large-scale networked services. In doing so, much of my work can be classified into three specific areas, detailed below: *(1) software infrastructure to support datacenter services, (2) network architectures to better manage the network's increased scale and complexity, and (3) systems for Internet content distribution*. Most of this work is in collaboration with graduate students, postdocs, and faculty colleagues, who, for brevity, are instead identified in the numbered citations on my vita.

1 Datacenter Infrastructure

Datacenter services have achieved scale through cheap hardware and designs that minimize shared state; my research has brought more rigorous guarantees about consistency, assurance, fault tolerance, and trust to this area.

First, a primer on today's architectures. For scale and availability, Internet services replicate software functionality across multiple machines to avoid dependencies on any single running process or piece of hardware. They also typically separate more stateless computational components from stateful storage, as the two can handle scaling and failures quite differently. For example, most modern web services deploy both a front-end tier of soft-state web servers and a back-end tier of databases or other storage systems. Then, web servers can be easily spun up or down as demand changes, and their crashes do not affect any durable state. Such a design places significant importance on the correctness and availability of the back-end storage systems, however. Much of my work on datacenter infrastructure has focused on these properties, and address the following problems:

1. Achieving stronger consistency in scalable storage systems, particularly when data is distantly replicated;
2. Introducing system-wide performance isolation and service assurance in multi-tenant datacenters;
3. Masking many types of failures and security violations, both due to malicious faults or in legacy systems;
4. Reducing trust in cloud platforms, and instead relying on technical means to provide privacy and integrity for cloud-hosted data and services; and
5. Offering new techniques of advanced data analysis for "Big Data" stored in and across datacenters.

This section is organized around these problems. Some of this work was funded by an NSF CAREER award, through which I recently received the Presidential Early Career Award for Scientists and Engineers (PECASE).

1.1 Stronger consistency for geo-replicated services

As distributed systems have grown larger and more distributed, designers have often sacrificed stronger consistency semantics due to their perceived cost and lack of scalability. Beyond the performance implications of wide-area transactions, systems offering strong consistency (linearizability) forgo availability when a sufficient number of sites are partitioned from one another. Yet this tradeoff is fundamental (formalized in the CAP Theorem), which has led most systems to instead embrace very weak "eventual" consistency models to ensure availability and low latency. Yet stronger consistency guarantees make systems easier for a programmer to reason about and provide users with more expected behavior (e.g., "what you write is what you'll subsequently get" or "the system preserves the order of your operations"). The move of ever-more dynamic and interactive applications to the cloud portends a shift in service requirements, in which losing data or exposing operations out-of-order may not be acceptable.

One approach we took to reduce the cost of strongly-consistent replicated state machines is to design protocols that are optimized for common datacenter workloads. For example, the CRAQ system [24] greatly improved read throughput and latency for read-heavy workloads. (Indeed, recent data published about Facebook's Tao caching system suggests read-to-write ratios around 500:1.) CRAQ's design allows any node in a replication chain to handle reads, as opposed to a single primary node (as in traditional primary-backup systems and its Chain Replication predecessor). This capability is particularly useful for chains stretching *across* datacenters, as many reads can be handled locally, or at worse require concise metadata to be transmitted across the wide area. Still, CRAQ does not circumvent the fundamental tradeoffs inherent to linearizable systems: Its operations during times of read-write contention involve wide-area replicas, and it sacrifices liveness when a sufficient number of nodes are unavailable or partitioned from one another.

Our more recent research *pushes the limits as to what transactional and consistency guarantees are possible more generally, while still preserving high availabilty, scalability, and low latency.* We designed distributed protocols and working prototypes that show how to achieve causal consistency,[1] as well as read- and write-only transactions, in such settings. Using these new protocols, all client requests can be satisfied in the local datacenter in which they arise; data is replicated only asynchronously between datacenters. But our key contribution is scalability. While it has been known since the 1970s how to ensure such consistency between operations serialized to a single server, our new protocols enforce causal dependencies between objects spread across an entire cluster, and rigorously enforce such causal orderings at all sites world-wide.

More specifically, rather than implicitly tracking ordering via logs or version vectors, our COPS key-value storage system [12] explicit adds metadata to stored values that precisely track dependencies. When a new value is replicated, the remote datacenter checks whether its causal dependencies on keys are satisfied locally before exposing the write. Such tracking also enables read-only transactions that obtain a consistent snapshot of multiple keys (across servers) without locking or blocking. Unfortunately, when a datacenter fails or is partitioned, the amount of this stored metadata can grow exponentially in the worst case.

We improved this result in our subsequent Eiger system [1]. In Eiger, client requests continue to be satisfied locally without blocking, the system supports causally-consistent read *and* write transactions efficiently, and it provides richer data model abstractions such as column families, counter columns, and secondary indexes (much like Google's BigTable, but now with scalable consistency properties). One of our key technical insight was to track dependencies on potentially complex operations, as opposed to COPS's explicit checks on data values, which also led to new transaction algorithms based on a novel use of logical time. The system's new mechanisms significantly reduce the amount of metadata and verification needed (and thus avoid COPS's potential metadata explosion).

1.2 Service assurance in shared clouds

Cloud platforms like Amazon Web Services have given rise to new business models, in which third-party "tenants" deploy their services within the cloud provider's datacenters and pay only for the incremental resources used. While shared storage enjoys wide adoption in commercial clouds, most of these systems provide weak performance isolation between tenants, if at all. This has inhibited adoption; customers are concerned that other, potentially misbehaving, tenants may overwhelm the shared resources.

Instead, we have developed systems that *rigorously enforce resource guarantees across an entire shared storage cluster, while still ensuring high utilization and avoiding strong assumptions about workload characteristics*. In particular, these mechanisms [5] ensure performance isolation and fairness even as workload demands are highly dynamic, skewed, or bottlenecked by different server resources. We did so by leveraging both theoretical

[1]More technically, the systems provide the stronger "causal consistency with convergent conflict handling" and can easily be adapted to provide real-time causal (RTC) consistency; it has recently been proven that no consistency model strictly stronger than RTC is possible under these constraints.

and low-level systems mechanisms. At a high-level, we decomposed our global optimization problem seeking max-min fairness into multiple mechanisms (based on the primal-allocation method of distributed convex optimization [2]): partition placement, weight allocation and multilateral weight swapping, congestion-driven replica selection, and weighted fair queuing extended to resource vectors and dominant resource fairness. These mechanisms operate on different timescales and with different levels of system-wide visibility. Yet, the efficient implementation of these mechanisms were driven by systems concerns, e.g., lock-free data structures to leverage local multi-core concurrency, which in turn required novel mechanisms for fair resource accounting and enforcement.

1.3 Fault-tolerant cloud services

As the number of machines comprising a service grows large, failures become the norm. It's standard procedure today to monitor crashes and to reboot processes or machines for recovery. However, many types of faults can be more subtle than simple crashes, including malicious attacks against vulnerable programs, and even crashes can affect a server's immediate clients. Traditionally, handling such faults is expensive and not scalable, and requires comprehensive changes to end-point applications. Instead, we have developed a number of mechanisms that *handle malicious faults with low overhead, scale to larger systems, mask certain faults even from legacy clients, and isolate the effect of vulnerable programs on larger services.*

Byzantine fault tolerance. To handle arbitrary failures, the distributed-systems community developed Byzantine fault-tolerant (BFT) replicated state machines. This general approach uses consensus protocols, such that if no more than f servers are faulty (typically out of $3f+1$ total), the system can maintain both safety and liveness. BFT replication costs a significant overhead in throughput, however, which has inhibited its use. Our Prophecy system [22] leveraged the read-heavy nature of cloud workloads—and slightly weakened consistency guarantees—to achieve throughput that is almost identical to unreplicated systems, and multiple times that of traditional BFT protocols.

To enable the use of BFT protocols in large-scale applications, we also developed mechanisms that *scale out* BFT, i.e., by safely composing many small BFT groups into a large-scale system [41][10]. In such settings, however, the adversary can adaptively rejoin its colluding nodes, in an attempt to colocate them in the same group and break the f-threshold assumption. Our main technical result, a group partitioning protocol that uses a "commensal" cuckoo hashing rule, ensures a much more balanced distribution of faulty nodes across groups.

Transparent crash recovery. While such consensus protocols provide high availability and prevent the corruption of replicated state, legacy users interacting with front-end servers (e.g., via a web browser) may observe crashes through connection time-outs, truncated downloads, or error messages. Instead, our TRODS system [16] transparently recovers connections during static content delivery, which comprises the majority of HTTP (and Internet) traffic. The key insight in TRODS is its use of cross-layer visibility and control: it carefully derives reliable storage for application-level state from the mechanics of the transport layer. This state is used to reconstruct content-delivery sessions, which are then transparently spliced into legacy end-points' ongoing connections.

Isolating vulnerabilities. Another major limitation of BFT protocols is that they assume servers fail independently, which is ill-matched to the software monoculture and single administrative control of datacenter services. We have recently explored techniques by which vulnerable components in networked services can be automatically isolated, so that even successful attacks that permit arbitrary code execution are limited in their capabilities. In particular, the Passe system [61] automatically splits previously single-process web applications into sandboxed processes (one per "view"), and then limits the types of queries each (potentially compromised) component can make to shared storage. While previous approaches like Decentralized Information Flow Control (DIFC) sought to provide security guarantees by explicitly labeling and tracking data, Passe provides similar guarantees by instead enforcing *integrity* constraints on database queries. Further, rather than requiring developers to specify such constraints explicitly, Passe *infers* these constraints, and thus the underlying security policies, from unmodified applications (currently those written in the Django web framework). It currently does so through a testing phase that uses a combination of trace analysis and object taint tracking, although we are exploring several richer forms of policy inference—inferring each view's finite state machine, using *limited* code annotation or policy invariants, etc.—to better model programmer intent and generate more precise integrity constraints on database access.

1.4 Untrusted cloud services

While cloud deployments are attractive for both economic and technical reasons, their benefits come at the cost of having to trust the cloud provider. Private data can be leaked to malicious parties or turned over to government agencies. A malicious or compromised cloud provider can corrupt users' data or even equivocate, showing different users divergent views of the system's state. In fact, bloggers claim that Sina Weibo, a Chinese microblogging service, actively uses such equivocation as a form of surreptitious censorship. Such security concerns have also inhibited adoption by corporate and government organizations.

This tradeoff is not fundamental: we have built systems that use a centralized cloud provider without trusting it with the privacy or integrity of users' data. The provider's servers see only encrypted data and cannot deviate from correct execution without detection. The cloud servers primarily act as a storage and ordering service, with most application logic pushed to the client (arguably not unlike many Web services today that heavily leverage Javascript executing in the browser). Our methods are general, and we have applied them to both group-collaboration applications (such as online word processing like Google Docs) and social-networking services [6] (like Facebook or Twitter).

The former group-collaboration system, SPORC [17], supports the concurrent editing of shared data, disconnected operation, and dynamic access control. Conceptually, SPORC unifies the benefits of *operational transformation* and *fork*-consistency* protocols: the former provides a means to concurrently execute lock-free operations that converge to a common state, which can also be repurposed to undo, replay, and merge any (encrypted) state of clients that had been forked by a malicious server.

The latter system, Frientegrity [6], focuses on the challenge of scale in order to address the needs of large social networking services. Users may have hundreds of friends (or orders of magnitude more friends-of-friends or followers). On the other hand, users are mainly interested in the latest updates to objects, not in the thousands of preceding ones.

Frientegrity introduces a novel method for detecting provider equivocation in which clients collaborate to verify correctness, as well as access-control mechanisms that scale efficiently (logarithmically) with the number of friends.

1.5 New techniques for large-scale data analysis

The above systems largely focus on managing stateful services, not on analyzing their potentially voluminous data. The past decade has seen a massive increase in the scale of this "Big Data," and, in reaction, industry and academia have developed data-processing systems that run across many, even thousands, of machines. One of the most popular of such systems, Google's MapReduce and its open-source Hadoop clone, runs embarrassingly parallel tasks as a series of barrier-synchronized jobs within a datacenter. While many researchers (and companies) focus on improving MapReduce performance, *my group has been exploring new forms of scalable data analysis that are infeasible today.*

First, designed for an important class of machine learning and optimization applications (e.g., stochastic gradient descent or LDA), we have been developing a system for incremental, *asynchronous* computation [69]. In these applications, computational subproblems may have wildly different execution times, yet many passes over data are needed for convergence. As such, they are ill-suited for both distributed shared-memory systems and barrier-synchronized frameworks. Instead, our system leverages update semantics found in these applications: namely, the system only needs to manage shared state that is both *advanceable* (state updates all make progress towards a shared computational goal) and *restartable* (repeated computation on the same data preserves this property).

Second, because network backhaul to a single datacenter falls far short of scaling with data generation rates, we have begun developing a system for the wide-area processing of both real-time streams and OLAP-style historical analysis [70]. Challenges in this problem space, which may involve multiple datacenters or even more constrained edge devices, abound: query optimization based on operator semantics, query adaptation due to dynamic network resources, aggregation and approximation based on system conditions, and in-network summarization data structures for historical analysis, among others. While there is a long history of research in stream processing and OLAP databases, few if any combine these mechanisms for wide-area, heterogeneous settings.

2 Network Architecture and Programmability

Now that Internet services have grown large and complex, their demands on the network have increased correspondingly. Unlike the huge innovation and evolution that has occurred in applications and services, however, network architectures and protocols have remained relatively static for decades. Network hardware vendors (e.g., Cisco and Juniper) serve as gatekeepers to new functionality, and a lack of interoperability further impedes innovation. In fact, the research community had taken to calling the current network "ossified," expressing its resistance to change.

Yet, most researchers accept the premise that the current layers must remain unchanged. They do not question the existing control/data-plane separation with decentralized control-plane protocols, and automatically adopt the host-centric stack abstractions of the 1960s. My research has challenged some of these basic assumptions, and derivatives have already begun to reshape the networking industry. This work includes:

1. Revisiting the traditional decentralized control/data architecture;
2. Programming networks with high-level, domain-specific languages, rather than configuration specifications; and
3. Rethinking the abstractions and layering on which hosts and services operate.

2.1 Software-defined network architectures

As a federation of autonomous systems that both cooperate and compete to route traffic, the Internet is well-suited for decentralized control. Control protocols within enterprise, carrier, and datacenter networks typically take decentralized approaches as well, even though they fall under a *single* administrative domain. In a line of research I helped develop during my Ph.D. studies, *we rethought how one manages a network and its switches, using centralizing logic and visibility for software-defined control.*

SANE [28] introduced a highly-secure network architecture built around the principle of least knowledge. A centralized controller performs admission control for all flows by granting (onion-encrypted) capabilities that enable circuit establishment for approved communication. Ethane [26][23] generalized this design both to operate in legacy Ethernet environments and to focus on flow management more broadly, rather than solely on security applications. Ethane allowed a network operator to define network-wide, fine-grained policies on higher-level principles (e.g., users, groups, and application protocols, rather than IP addresses and VLANs), and then enforce these policies directly in switches. Ethane reduced network switches to their minimum—simple flow-based forwarding planes (FIBs)—and moved all control logic to a centralized controller that installed forwarding rules into the switches on demand.

Ethane's architecture, protocols, and prototype developed into the OpenFlow and software-defined networking (SDN) architectures (www.openflow.org). *OpenFlow and SDN have had a major, rapid impact on the networking industry*: OpenFlow is now being standardized through the newly-formed Open Networking Foundation, at least fourteen switch vendors interoperably support OF in hardware, a software OpenFlow switch is in the mainline Linux kernel, and companies like Google use OpenFlow to manage their backbone and datacenter networks.

More recently, DIFANE [18] challenged whether reactive flow installation requires the centralized controller's involvement, as this introduces scalability and fault-tolerance concerns. Instead, DIFANE keeps all traffic in the fast data path. Rather than falling back to the controller when no installed (cached) flow rules match a packet, switches selectively direct packets through other intermediate switches that cooperate to store the entire ruleset. DIFANE then relegates the controller to the simpler task of partitioning these (potentially overlapping) rules over the switches.

2.2 Programming software-defined networks

OpenFlow and its centralized control architecture have become a platform for network programmability, helping to manage interrelated services including routing, traffic monitoring, load balancing, and access control. Unfortunately, the languages used to program today's networks lack modern features: They are usually defined at the low level of abstraction supplied by the underlying hardware, and they fail to provide even rudimentary support for modular programming.

In a project led by programming-languages colleagues, *we have developed the first high-level programming language and runtime for OpenFlow and SDN.* The domain-specific Frenetic [44][15][39] provides high-level abstractions that still give programmers direct control over network devices, modular constructs that support composition, portability to different devices, and rigorous semantic foundations. The runtime handles many low-level packet-processing details for programmers, yet keeps traffic in the data path whenever possible.

In such software-defined architectures, one need not be limited to existing (layer-2 and -3) network abstractions. One ongoing project is developing a programming framework for *joint* network and end-host monitoring and control [43]. Another significant project introduces service abstractions as first-class network primitives, discussed next.

2.3 Service-centric networking

Internet services run on multiple servers in different locations, serving clients that are often mobile and multi-homed. This does not match well with today's network stack, designed for communication between fixed hosts with topology-dependent addresses. *Our work revisits the decades-old design of this stack—and the naming and software abstractions it exposes—to meet the needs of online services.* The centerpiece of our Serval architecture [9] is a new Service Access Layer (SAL) that sits above an *unmodified* network layer, and enables applications to communicate directly on service names. The SAL introduces a clean service-level control/data plane split for software-defined control of services and end-points.

With Serval, end-points can seamlessly change network addresses, migrate flows across interfaces, or establish additional flows for efficient and uninterrupted service access. By rethinking the layers in the stack, Serval can support a wide range of functionality in a coherent, clean architecture—including service-level anycast and load-balancing, mobility and migration, multi-path and multi-homing, and secure communication—many of which were previously handled by incomplete point solutions. But it does so without modifying the network layer (and thus operates across the legacy Internet without awkward, brittle tunneling), and it requires few to no changes to applications. Indeed, we placed significant importance on a design and implementation that supports incremental deployment; assuming a fully "clean-slate" Internet is a non-starter in practice.

Serval (www.serval-arch.org) is a major systems-building effort, and we have learned from the multiple prototypes we built over several years. The current prototype is more than 30,000 lines of C code and runs both as a loadable Linux kernel module and in user space (tested on Linux, Android, and Mac OS X). It has been used to build services both for datacenters and mobile clients, and its protocols have been formally verified for correctness [4]. Serval was partially funded by faculty awards from the ONR Young Investigator Program and DARPA Computer Science Study Group program, and the work has received significant interest from industry, including Cisco, Verizon, and AT&T.

3 Distributed Systems for Content Delivery

Much of my research on content delivery, performed before the rise of advertising-supported content hosting sites like YouTube, sought to "democratize" content distribution. I designed and deployed efficient, self-organizing architectures that made desired content widely available, regardless of the publishers' own resources. More recently, we brought greater rigor and insight to the design of such systems. This research includes:

1. Enabling cooperative web proxies to discover cached content efficiently;
2. Understanding how peers may be incentivized to contribute resources, and ensuring that their shared content is not corrupted;
3. Providing near-optimal wide-area server selection without centralized control; and
4. Addressing the challenges and opportunities afforded by 3D content and user-generated environments.

3.1 Peer-to-Peer and content distribution systems

Content distribution networks (CDNs) are not a new idea, but commercial architectures are bound to more static, manual deployments. Instead, my CoralCDN system [37] introduced self-organizing algorithms for indexing and clustering, which enabled proxies to coordinate their web caches dynamically in order to minimize load on origin web sites. CoralCDN's underlying scalable index [54] was based on a distributed hash table (DHT), but uniquely designed for finding nearby content and operating under overload conditions.

CoralCDN (www.coralcdn.org) *is a practical success*: it handles tens of millions of requests from several million unique users per day. I have operated it continuously since 2004 at hundreds of servers world-wide. CoralCDN's use is varied, from software downloads, to resurrecting overloaded pages, and to serving popular images and videos. It has been of particular use during times of disaster management (e.g., for sharing amateur videos during the 2004 Indian Ocean tsunami, and for caching overloaded websites linked from a special URL shortener developed during the 2011 Japanese earthquake and tsunami). Longitudinal experience operating CoralCDN for five years (now almost nine) uncovered a number of challenges and design lessons for CDNs, wide-area distributed systems, and shared deployment platforms [21].

3.2 Large-file distribution

While CoralCDN, like most CDNs, focused on whole-file distribution, the rise of peer-to-peer file-sharing systems like BitTorrent saw the adoption of multi-source transfer protocols, in which chunks of files are simultaneously downloaded from multiple peers. To ensure the availability of content, many of these systems employ schemes to incentivize users to share, such as BitTorrent' "tit-for-tat" protocol. In a more theoretical line of work [48][47][25], we employed *price theory* to study how peer demand can be efficiently matched to available supply. We also characterized the efficiency and robustness gains that can be enabled by price-based multilateral exchange [13], as opposed to bilateral exchange as in BitTorrent. (Namely, the multilateral equilibrium allocation is Pareto efficient, while bilateral exchange is not, although we also quantified those regimes in which bilateral exchange may still perform quite well.)

Open peer-to-peer systems for large-file distribution are also at risk from malicious users trying to disseminate incorrect blocks. Systems that split files directly into chunks, like BitTorrent, can use standard Merkle hash trees for chunk verification. When content is encoded with rateless erasure codes for performance and availability reasons, however, such mechanisms do not work. To secure such protocols, we introduced efficient block-verification mechanisms based on a homomorphic hash function [36]. We also developed a network file system in which cooperating clients can securely read cached blocks from each others' file caches [32].

3.3 Server selection and network measurement

When Internet systems are distributed across multiple locations world-wide, client performance is strongly tied to the careful selection of nearby, less-loaded datacenters or service replicas. Motivated from my experiences with CoralCDN, we designed and deployed server-selection systems based on self-organizing algorithms, eschewing traditional centralized visibility and control.

OASIS [29][77] devised efficient mechanisms to map the Internet topology cooperatively in order to determine geographic proximity. It leveraged the network vantages of nodes from participating services to perform network measurement, detected and disambiguated erroneous results, and represented locality stably across time and deployment changes. During its deployment from 2005 to 2009, it managed thousands of service replicas for more than a dozen distributed services.

Its successor, DONAR [19] (www.donardns.org), introduced more rigorous guarantees and more flexible load-balancing policies. Instead of the heuristics used by most systems, DONAR provides a constraint-based optimization that robustly decomposes the selection process into a decentralized algorithm. Of practical impact, we deployed DONAR in 2010 as the production DNS service for Measurement Lab, a global network measurement platform jointly operated by the PlanetLab Consortium, Google, and the New America Foundation. *DONAR handles hundreds of thousands of requests daily, including use by the FCC for its Consumer Broadband Testing.*

In the process of developing and monitoring these systems, we have performed a number of Internet measurement studies, including those to characterize CDN flash crowds [11], to clarify the interplay between IP prefix geolocation and aggregation [31], and to remotely identify transparent edge technologies like NATs and proxies and to understand their effects on IP-based "intelligence" [27]. Technology developed in the latter study was acquired and commercialized by Quova in 2006, which was subsequently acquired by Neustar.

3.4 Systems for 3D content

More recently, we have explored the feasibility of building large, complex, and seamless metaverses: three-dimensional virtual worlds where anyone can add and script new objects. Yet in metaverses like Second Life and games like World of Warcraft, users can see and interact with only a tiny region around them. Current metaverses impose this distance restriction for scalability, as it avoids the need to share appreciable state between regions, which may be managed by different servers.

In contrast, our Sirikata metaverse platform [8][75] (www.sirikata.com) scales to large, complex worlds, even as it avoids such restrictions. It does so by leveraging properties of the real world in its core systems: for example, novel distributed data structures prioritize queries and communication between objects (and aggregates) based on their visible sizes (i.e., solid angles) [65].

Three-dimensional content (textures and meshes) also introduces novel problems for efficient delivery. First, while users should be able to insert any 3D content into the world, arbitrary content is often not optimized for real-time rendering. Our content-conditioning pipeline robustly transforms any 3D content into a progressively encoded format in an unsupervised fashion, while minimizing the loss of quality [7] (www.open3dhub.com). Second, while browsers typically download web objects sequentially, a metaverse operating on progressively encoded content has much greater flexibility in scheduling transfers. Our scheduling algorithm carefully optimizes the choice of base object, texture resolution, or mesh vertices to maximize visual fidelity [64].

Security and Privacy

Beyond the major themes described above, I have pursued a number of systems projects related to anti-censorship and anonymous communication [60][59][56][38][42], anti-spam based on social proximity [30][49], and privacy-preserving, collaborative data aggregation [20]. I also worked on various cryptographic protocols, particularly in the area of secure multi-party computation (e.g., private set intersection [35][72] and private keyword search [33]). One significant result of cheap storage and scalable cloud architectures has been the ability for companies (and governments) to capture, store, and analyze vast troves of data. *In a world where too much information is collected and shared, much of my security research addresses how users can minimize what information can be learned about them, or how parties can cooperate to learn joint information without needing to share their raw data.*

Final Thoughts

My research career has coincided with massive changes in the use of computing, which escaped from scientific and back-office applications to become entwined in all aspects of modern life. Computer systems have grown to be found everywhere; Internet services have grown to be accessible anywhere.

Such change was accompanied (or perhaps enabled) by a significant shift in the way we provide these services: from the special-purpose, tightly coupled hardware monolith, to a set of loosely coupled software components running on commoditized, sometimes virtualized, machines. And one of the last major bastions of highly specialized, vendor-controlled hardware—networking—is beginning to witness similar disruption. Software has become *king*, and distributed systems have grown in size by orders of magnitude.

But building complex systems is difficult, and our community continues to struggle with challenges of reliability, security, performance, scale, and efficiency. I believe that my research—through its novel algorithms, protocols, system architectures, and production deployments—has helped further the state-of-the-art for Internet services that operate at scale, across networks, and with a high degree of robustness and performance. And that is a worthwhile endeavor, indeed.