Taming Wireless Fluctuations by Predictive Queuing Using a Sparse-Coding Link-State Model

Stephen J. Tarsa
Harvard University
Cambridge, MA

Marcus Comiter
Harvard University
Cambridge, MA

Michael B. Crouse
Harvard University
Cambridge, MA

Bradley McDanel
Harvard University
Cambridge, MA

H. T. Kung
Harvard University
Cambridge, MA

ABSTRACT

We introduce State-Informed Link-Layer Queuing (SILQ), a system that models, predicts, and avoids packet delivery failures caused by temporary wireless outages in everyday scenarios. By stabilizing connections in adverse link conditions, SILQ boosts throughput and reduces performance variation for network applications, for example by preventing unnecessary TCP timeouts due to dead zones, elevators, and subway tunnels. SILQ makes predictions in real-time by actively probing links, matching measurements to an overcomplete dictionary of patterns learned offline, and classifying the resulting sparse feature vectors to identify those that precede outages. We use a clustering method called *sparse coding* to build our data-driven link model, and show that it produces more variation-tolerant predictions than traditional loss-rate, location-based, or Markov chain techniques.

We present extensive data collection and field-validation of SILQ in airborne, indoor, and urban scenarios of practical interest. We show how offline unsupervised learning discovers link-state patterns that are stable across diverse networks and signal-propagation environments. Using these canonical primitives, we train outage predictors for 802.11 (Wi-Fi) and 3G cellular networks to demonstrate TCP throughput gains of 4x with off-the-shelf mobile devices. SILQ addresses delivery failures solely at the link layer, requires no new hardware, and upholds the end-to-end design principle to enable easy integration across applications, devices, and networks.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless networking, 802.11 (Wi-Fi), cellular data networks, TCP, sparse coding, data-driven model learning

1. INTRODUCTION

Wireless networks struggle to cope with link fluctuations in everyday scenarios. Any user who has accessed in-flight Wi-Fi, checked a smart phone on an elevator, or browsed the web on a train can attest to frustrating pockets of dismal throughput. Such disruptions result from the drastic changes in signal propagation that occur as users travel through complex environments with varying sources of multipath reflection, distances to base stations, line-of-sight occlusions, and interference. Generous provisioning of network infrastructure is the de-facto solution in practice, but more flexible and scalable strategies are needed [1].

Addressing delivery failure at the link layer is attractive because it shields higher layers of the network stack from transient link variations without changing radio hardware [2, 3]. When packet delivery is stable, applications and protocols like TCP realize higher throughput with lower variation, simplifying design and improving user experience [4]. However, predicting wireless link state at the scale of individual packets is difficult in real-world scenarios where steady-state assumptions like stable loss rates or location-dependence have limited efficacy. For example, we see in indoor experiments that the relationship between physical location and packet loss can change without warning if a coworker politely holds open a metal door, extending a link's range. These effects are common, especially in environments where construction predates pervasive wireless communication.

In response, we present State-Informed Link-Layer Queuing (SILQ), a system that uses data-driven learning to mitigate the destabilizing effects of wireless link loss. SILQ predicts link state at the 100ms time-scale to anticipate tempo-

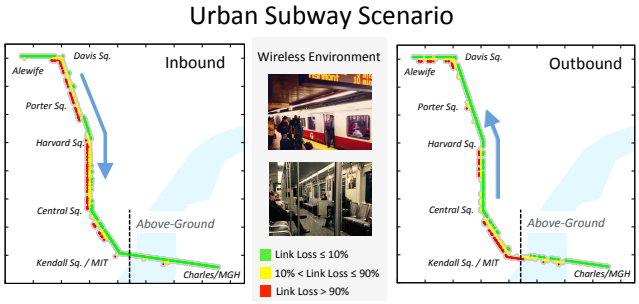

Figure 1: Packet loss rates over a 3G cellular network are shown on the subway in Boston, MA. Temporary dead zones in tunnels cause connection timeouts that leave links under-utilized, even when the train comes above-ground.

MobiHoc'15, June 22–25, 2015, Hangzhou, China.

http://dx.doi.org/10.1145/2746285.2746318.

rary outages and buffer packets accordingly. Real-time predictions are made by actively probing links, matching the resulting measurement sequences to an overcomplete dictionary of prominent patterns learned offline, and then classifying these *sparse feature vectors* to determine if they precede an outage. By preempting transmissions that are unlikely to succeed, SILQ reduces performance degradation caused when higher layers of the network stack react adversely to missing packets, for instance when TCP times out in a subway tunnel. Unlike prior cross-layer approaches to this problem, SILQ does not break connections in the middle of the network or modify transport protocols [4, 5, 6].

SILQ's prediction model improves upon past data-driven link models by expressing state as a linear combination of a few canonical packet-loss patterns, called features. This *sparse linear model* cuts away noise and restores missing data to improve statistical accuracy in the face of real-world data variations [7]. More-stable feature vectors can then be classified with a linear Support Vector Machine (SVM) to identify outages [8]. This method produces stable predictions looking further ahead in time than standard techniques like regression [9]. We demonstrate that this noise-, variation-, and deletion-tolerant model is computationally simple enough for real-time operation on mobile devices, and also supports the design flexibility to trade prediction accuracy for bandwidth and power improvements.

We build our predictive link model by first learning a dictionary of link-state features offline. Training data consists of UDP packet receptions recorded in three scenarios:

- An Unmanned Aerial Vehicle (UAV) flying over rural farmland
- Users walking throughout an active office building and riding an elevator
- Users traveling through a city on the subway

We show that features captured by sparse coding, a form of unsupervised clustering [10], reflect the effects of line-of-sight occlusions, radio range limitations, and interference. Furthermore, we show that features are often universal across rural, urban, indoor, outdoor, high mobility, and low mobility links. In contrast, we find that our supervised SVM outage-predictors are specific to an environment, though predictions remain accurate over days, weeks, and months.

This paper presents a full implementation of SILQ on off-the-shelf Linux and Android devices, and validates it in the field over 802.11b/g and 3G cellular networks. We compare our sparse-coding link model to traditional modeling approaches based on loss-rate and heuristics in terms of both statistical accuracy and network throughput under predictive queuing. Ultimately, we show that SILQ boosts throughput in an indoor office by 2x when used with Linux TCP, while achieving 4x lower variation. SILQ further improves throughput by 4x on the Boston subway where 3G coverage is spotty. Finally, we show that the power overhead of our approach can be reduced to only 4-5% above a baseline data connection by updating predictions less frequently when the link is in a steady state.

2. TEST SCENARIOS

We begin by describing our UAV, indoor office, and urban subway scenarios. In each environment, we collect initial link measurements by transmitting 66-byte (66B) UDP probe packets every 1ms from a mobile node to a nearby base station that logs sequence numbers. We detect lost probes from non-sequential receptions and construct contiguous streams of probe deliveries, called "traces." These high-granularity traces are used to build a link model in Section 4, though we introduce an adaptive probing mechanism in Section 5.1 to reduce SILQ's runtime probing overheads.

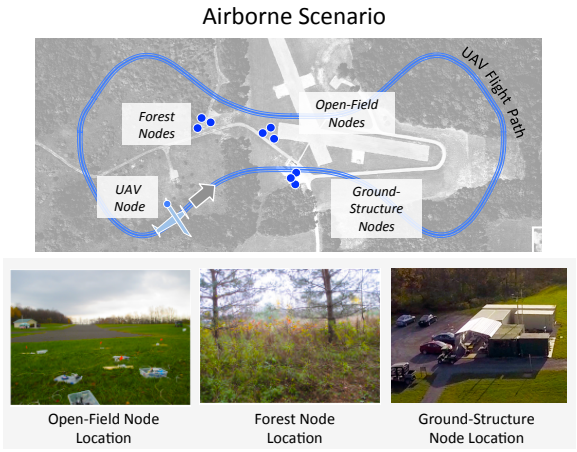

Figure 2: Our UAV's flight path over fields and farmland in upstate New York is shown. Ground receivers are placed in three locations to capture the link characteristics of an open field, dense forest, and parking lot with surrounding ground structures.

In our experiments, transmitters and receivers are all off-the-shelf mobile devices. For 802.11 measurements in the UAV and indoor office scenarios, we use an array of laptops, Mobile Internet Devices (MIDs), and BeagleBone Blacks depending on weight, power, and portability restrictions. Each accesses the wireless medium using a T-Link TL-WN727 USB dongle, and we disable hardware retransmissions by sending packets in broadcast mode. For subway experiments, we access the 3G cellular network with an Android smartphone or by tethering nodes to an Apple iPhone 6. In this case, we cannot disable ARQ with cell towers, though the effect is constant throughout data.

Given the speed and range of our UAV links, we transmit at 1Mbps modulation for best symbol resilience. In order to compare results across environments, we rate-limit links in the office and subway to 1Mbps. In practice, SILQ has been scaled to 20Mbps and deployed on commercial 802.11, 3G, and LTE cellular networks.

2.1 UAV Air-to-Ground Communication

Our UAV scenario consists of a fixed-wing SIG Rascal 110 flying a dumbell shaped pattern over an airfield in upstate New York on precipitation-free days, as shown in Figure 2. Autopilot maintains a consistent flight path to within 15m across laps, while keeping altitude steady at 100m and velocity at 20m/s. The property spans forest and farmland, contains fewer than 50 ground structures or cars at any time, and is free of 802.11 interference. Data collection is conducted over 802.11b with a single broadcast transmitter attached face-down to the UAV's wing.

Figure 3: Our walking tour throughout an active office is shown in blue. Links are attenuated by concrete and brick, and obstructed by fire-proof doors and an elevator car.

Ground receivers are placed in identical orientations in three groups throughout the property to capture different signal-propagation characteristics:

1. *Field* nodes at the end of the runway have clean line-of-sight to the UAV during most of the flight. For them, packet loss is driven primarily by radio range.

2. *Ground structure* nodes are placed in a parking lot partially surrounded by several 5m-tall metal buildings. Between 4 and 6 cars are parked within 10m of nodes, and vehicle positions change across flights. These structures create line-of-sight occlusions and multipath reflections that affect packet delivery depending on geometry and UAV position.

3. *Forest* nodes are placed in a grove of fir trees at the end of the runway. They capture similar link conditions to field nodes, though uneven ground, trees, branches, and leaves affect line-of-sight transmission.

This UAV scenario is of both experimental and operational interest. First, the simple interference-free environment is ideal for conducting controlled wireless experiments to serve as building blocks for more complicated scenarios. Second, UAV wireless communication has commercial applications in bridge and crop inspection, rural Internet relay, package delivery, and a myriad of military scenarios.

2.2 Indoor Office

Our indoor office scenario is shown in Figure 3. A user carries a laptop throughout an office building while transmitting data to an 802.11 access point in a second-floor office. For each experiment, the user repeats a walking tour that crosses a footbridge to a nearby building and then returns for an elevator ride down three floors to the basement.

All experiments are conducted during high-traffic daytime hours when the building is used by several hundred students and faculty. We observe strong fluctuations in link quality due to signal attenuation through brick and concrete, as well as blockages from fire-proof doors and the elevator car. We also observe that signals are powerful enough to reach several floors to the basement, but that links can be blocked at ranges as short as 15m should fire doors occlude transmission. For this scenario, our data includes cross-channel interference from the building's 802.11 network.

2.3 Subway

Our subway scenario is shown in Figure 1. A user travels between Cambridge and Boston, MA in a train car that passes both below and above ground. Train speed varies depending on track conditions, and precise positioning is unknown within tunnels. The user transmits data from a mobile device to a server accessed via a first-hop 3G cellular connection and subsequent hops over wired Internet. We observe that round trip delays average 150ms and fluctuate within a manageable distribution. Queuing delays cause UDP probe packets to arrive in bursts of roughly consistent length and spacing.

As illustrated in Figure 1, we observe many regions of poor reception, but also extended regions of excellent reception when 3G repeaters are nearby or the subway crosses an above-ground bridge. This scenario is completely uncontrolled and is used to validate SILQ's methodology in an everyday situation of particular aggravation to the authors.

3. PREDICTIVE LINK-STATE MODELING

In order to implement predictive queueing, we first build a model of wireless packet delivery for our three scenarios. In the literature, link models typically reflect two approaches: physical models that use signal propagation to explain packet loss, and data-driven models that mathematically capture those effects with statistics like correlation between packet receptions. The former group includes distance-based attenuation, two-ray interference, and geometric occlusion models [11, 12, 13], while the latter includes loss-rate and Markov-chain models that reproduce distributions of consecutive receptions [14, 15, 16]. Though physics-driven methods are intuitive, they require detailed environment-specific information as inputs. On the other hand, generic data-driven models can require training datasets that grow exponentially with temporal scale.

These two extremes are sometimes blended by location-based statistical models [17, 18]. However, we find that location information is generally a poor predictor of individual packet losses. For example, we observe in our UAV experiments that slight changes to antenna orientation drive major swings in link loss. We also see that the location of a single car close to ground-structure nodes causes drastic changes in packet loss behavior due to multipath reflection. Indoors, we see that loss characteristics change when doors are closed or metal objects are moved within offices. And in the subway, location information is not even available once the train descends underground. For these reasons, we *purposefully* eschew location information in pursuit of a more accurate, general packet loss model.

Our data-driven link model combines unsupervised sparse coding, a form of clustering, with supervised Support Vector Machine (SVM) classification. This approach is closely related to state-of-the-art learning systems for vision and speech recognition that capture complicated features while tolerating non-Gaussian data variations [19]. Though ours is the first sparse-feature model for wireless packet transmission to our knowledge, other learning techniques are gaining

traction in networking scenarios. At the transport layer, game-theoretic simulations can discover TCP protocols tailored to network conditions [20]. At the physical layer, learning has been used to adapt forward-error-correction (FEC) to congestion in simulation [21]. Our approach stands out both by our use of sparse feature representation to cope with real-world measurement artifacts, as well our emphasis on live, online operation aboard mobile devices.

3.1 A Sparse Coding Link Model

Sparse coding is an unsupervised clustering method that solves for an overcomplete matrix of recurring patterns in training data. We call that matrix a *dictionary* and its column vectors *features* [10, 22]. Under this framework, a data vector can be approximated as a linear combination of features. By constraining the number of features used, approximations are represented by *sparse* coefficient vectors with only a few non-zero entries. For data with noise or non-Gaussian variations like deletions, sparse approximations latch measurements to exemplar patterns, truncating weakly expressed information and restoring missing values. This process improves variation-tolerance in subsequent statistical analysis.

3.1.1 Offline Dictionary Training

We learn dictionaries offline from a large set of link traces, effectively exploiting past data to improve statistical accuracy for future data. Several formulations for dictionary training exist [23]; we use an optimization method based on the Least Absolute Shrinkage and Selection Operator (LASSO) that relaxes sparse coding's constraint on the number of non-zero feature coefficients by instead penalizing the sum of coefficient magnitudes [24]:

$$\min_{\alpha,D} \quad \frac{1}{2}||x - D\alpha||_2^2 + \lambda||\alpha||_1 \tag{1}$$

where D is a $w \times m$ dictionary containing m features of length w, x a normalized binary measurement vector representing the outcomes of w back-to-back probe transmissions, α its sparse feature representation, and λ a tunable positive value. To implement dictionary training, we segment field traces into T windows of w measurements x_t for $t = 1...T$. This objective produces qualitatively similar results to sparse coding but is convex, yielding stable, accurate solutions for D.

3.1.2 Online Measurement Encoding

While LASSO is effective for offline dictionary learning, it is too computationally intensive to compute sparse codes in real-time on mobile devices. Therefore, we use a D found by LASSO and solve for α online according to:

$$\min_{\alpha,D} \quad ||x - D\alpha||_2^2 \text{ s.t. } ||\alpha||_0 \leq k \tag{2}$$

with k a hard sparsity threshold. This alternative formulation can be greedily solved by Matching Pursuit (MP) [25] using a series of inner-product tests that iteratively select k features to reduce a residual. MP has $O(mw)$ complexity and we show in Section 6.4 that feature encoding is a negligible part of SILQ's running time.

Our sparse-feature-based link model is related to state models like Markov chains in the literature [14, 16]. The process of learning a dictionary by clustering can be thought of as a data-driven method for reducing the space of possible states in the model. For example, if we train a dictionary with many features and a small penalty parameter (e.g. $m \approx 200$ and $\lambda \approx 0.1$), then D captures *templates* of frequently occurring link states. New link measurements are then matched to the closest template by setting $k = 1$. However, if the number of dictionary features is chosen to be small and lambda is chosen to be large (e.g. $m \approx 20$ and $\lambda \approx 4$), then training will extract a small number of the most prominent patterns in each training example. This produces a set of more-general features that can be combined to enumerate observed link states, for example by encoding with $k = 8$. We call these features *primitives*. In Section 4, we will show that these latter linear-combinational models generalize well across networks and environments, addressing a key limitation of traditional wireless link models.

3.2 Outage Prediction With SVMs

Using learned link primitives, we can build a predictive statistical model for all sorts of link-layer events that affect network performance. In this work, we use bulk data transport via TCP as an application vehicle to demonstrate SILQ's gains and define the prediction target to be *any upcoming sequence of 192 probe measurements containing a single delivery failure*. This represents back-to-back transmission failures of 1500B MTU data packets sent at 1Mbps with the Linux default 7 ARQ retries, assuming a round-trip time less than 100ms. Since most TCP implementations can recover quickly from a single packet drop using fast-retransmit, we instead try to detect these two-packet outages. We note that video streaming and transaction processing are other candidate applications for SILQ with potentially different prediction targets.

To implement binary event prediction, we use a linear SVM classifier, which finds a separating plane between training data belonging to two classes, designated by class *labels*; later, new data vectors are compared to the separating plane using an inner-product and threshold test to predict their class membership. Since SVM training data in our application requires both sparse feature vectors and corresponding labels, it is considered a supervised technique. To derive labels y_t for our data, we annotate a subset of traces from each environment, setting $y_t = 1$ when an outage followed the measurement vector in the trace and $y_t = 0$ otherwise. In Section 4.2, we show how to optimize over SVM configuration parameters.

4. TRAINING MODELS

4.1 Learned Link Primitives

We first train dictionaries on field-collected traces using LASSO and examine the link primitives discovered in each of our scenarios. We learn separate dictionaries of size $m = 20$ with $\lambda = 4$ from each type of receiver: UAV field nodes, UAV forest nodes, UAV ground-structure nodes, indoor office nodes, and subway nodes. Figure 4 shows several features found often in each trace, with dips reflecting patterns of delivery failure and peaks capturing delivery successes.

We see that primitives learned across UAV ground receiver types are similar. Taken together, they capture examples of abrupt changes to the link, gradual transitions, intermittent packet drops, and temporary outages lasting 10's of milliseconds. When we examine the distributions of

nonzero coefficients in each trace's sparse representations, we find that field and forest nodes more often map to long runs of delivery successes or smooth gradual changes in link quality. This aligns with our experiment configuration, in which these nodes have the best connectivity, are far from any structures that can suddenly occlude line-of-sight transmission, and are affected mostly by radio range limitations. In contrast, the coefficients computed for ground-structure nodes more-often map to abrupt transitions as buildings block line-of-sight until the UAV is immediately overhead.

Primitives learned in the office environment bear a striking resemblance to those learned from UAV receivers. We see a mixture of temporary outages affecting strong links, as well as abrupt on-to-off and off-to-on transitions. These links exhibit a greater degree of noisy variation than UAV nodes, possibly due to 802.11 cross-channel interference from the building's wireless network.

Subway atoms reflect the same general shapes, but prominently exhibit regular 10ms-long oscillations in delivery. According to log files, average probe inter-arrival times over the 3G network display 10ms bursts of back-to-back deliveries, likely due to queueing delays. This means that unsupervised learning captures primitives reflecting both wireless-link and network effects.

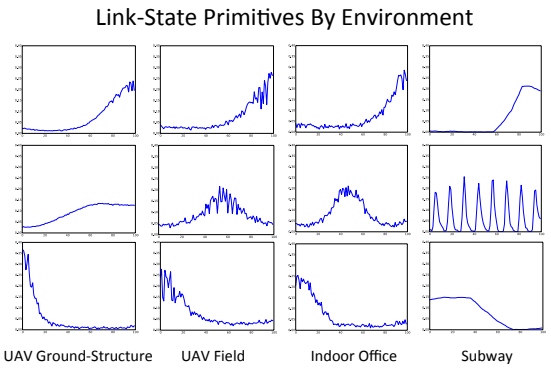

Figure 4: Common link primitives learned from different environments using LASSO with $m = 20$ and $\lambda = 4$ are shown. They capture fast and slow link transitions as well as bursts of delivery successes.

4.1.1 *Separating Network Effects*

As discussed in Section 3.1, sparse feature models can be configured either to match link states to a single template, or to express them as linear combinations of primitives. The practical result of this choice can be seen by comparing the primitives in Figure 4 to the template in Figure 5 learned from subway data using $m = 200$ and $\lambda = 0.1$. We see that templates simultaneously represent the concurrent effects of wireless link loss *and* network queuing delays. This contrasts with models based on linear combinations of primitives that automatically separate these effects.

This observation highlights the power of applying sparse coding to link modeling: our approach naturally expresses complicated network measurements in a simple, well-separated vector of high-level information – statistics can then be computed relative to this meta-information. In addition to supporting stable, intuitive statistical models, our method uses data-driven learning to identify link states that generalize beyond just their training data, reducing the need for extensive data collection in new scenarios.

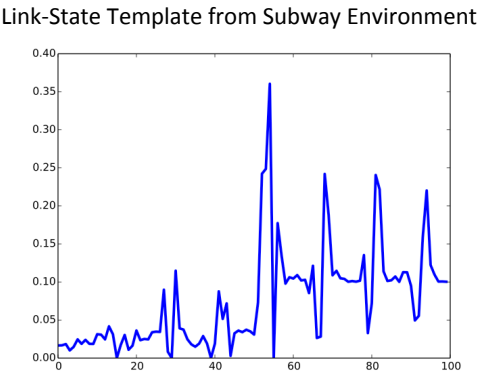

Figure 5: A link-state template learned from subway data is shown. The effects of link loss and bursts due to network delays are captured together, contrasting with Figure 4 where they are separated into different primitives.

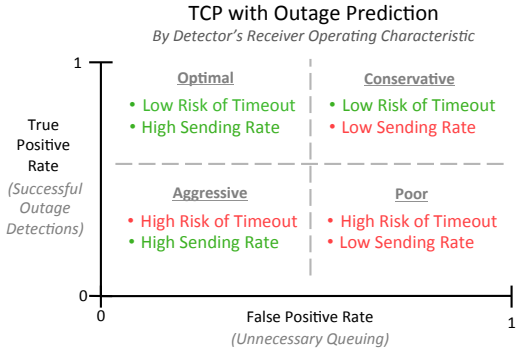

Figure 6: The effect of outage-detection sensitivity on TCP is illustrated. Detectors that forward data aggressively run a high risk of timeout. Those that forward conservatively rarely time out, but often under-utilize the link.

4.2 Tuning an SVM Predictor for Temporary Link Outages

We next show how to tune a linear SVM to predict outages from sparse-coded link measurements. Fundamentally, SVMs separate classes of data by solving for a max-margin separator between training examples of each class. When used for detection, this allows us to obtain a tradeoff between true positive and false positive rates by shifting the SVM separator toward one class or another.

Within SILQ, a predictor's sensitivity ultimately affects realized network performance. Figure 6 illustrates this relationship in the context of an outage predictor and TCP throughput. Predictors tuned to forward packets aggressively raise few alarms, achieving low false positives but also low true positives. The result is a fast sending rate but a high risk of TCP timeout when outages are not well-predicted. At the other end of the spectrum are predictors tuned to forward data conservatively by raising many alarms, leading to

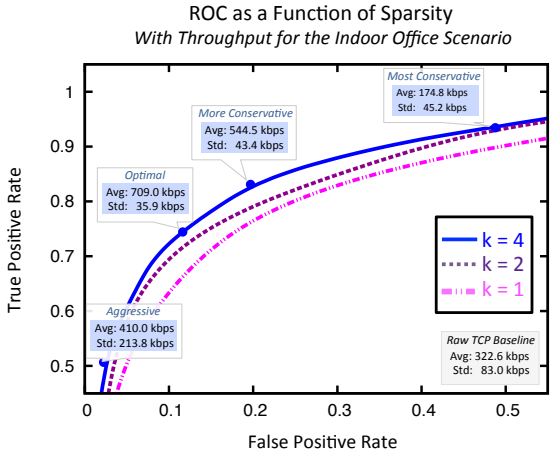

Figure 7: Throughput is shown at various points on our outage detector's ROC profile, empirically demonstrating the relationship described in Figure 6 in our indoor office scenario. We also see that training parameters such as encoding sparsity k affect the ROC profile of our outage detector.

a high true positive rate but also many false positives. These detectors rarely let timeouts happen, but severely underutilize links in good conditions.

In Figure 7, we show the empirical effect that prediction sensitivity has on throughput, measured in our indoor office scenario with the methodology of Section 6.2. We see that aggressive models achieve poor average throughput with a high standard deviation – while they opportunistically grab all available sending opportunities, they also often time out and cause throughput to drop to zero for a long period of time. However, overly conservative models end up realizing *lower* throughput since they fail to send often enough when the link is usable.

We also see in Figure 7 that optimizing over sparse coding parameters like m, λ, and k changes the resulting SVM predictor's ROC profile. For example, if we set $m = 20$, but $k = 2$, then the detector is strictly *worse* than setting $k = 4$.

Guided by these results, we reason that the best link model in terms of TCP throughput should maximize correct outage predictions while ensuring plenty of sending opportunities. We implement this by choosing the model that maximizes outage recall, provided non-outage (i.e. sending opportunity) recall is above a threshold of 0.75. Varying m, λ, k, and the SVM's test threshold, we train dictionaries and SVMs while holding out a subset of traces for evaluation to obtain $m = 20$, $\lambda = 4$, and $k = 4$ as our best configuration. Examining our optimized link model, we notice that the SVM heavily weights three particular dictionary primitives, suggesting that TCP throughput is highly sensitive to a small number of link states.

4.3 Transferring Features

While some link-state primitives may be unique to a particular environment, Section 4.1 showed that similar features are found among scenarios. Intuitively, these should represent common drivers of link loss like range effects in large open environments or abrupt transitions due to occlusions. We next empirically verify that learning captures canonical link characteristics that are common across diverse environments.

For Figure 8, we learn primitives from a single scenario and use them to compute sparse feature representations and train outage predictors in a different scenario. We not only see that primitives generalize well across environments, but that some amount of data diversity actually improves outage recall, possibly serving as a defense against overfitting. We note that this trend does *not* hold when we use a template model – for example, we find that templates from the office scenario do not accurately characterize the queuing effects of the subway, hurting recall.

We also found that SVM separators do not port as well as dictionaries across scenarios, since the precise relationships between link states over time are environment-dependent. For example, a fast subway train may produce shorter temporal correlation between a degrading link and an upcoming outage than in the office scenario. However, we saw that SVMs trained on months-old data still provided good performance in the field, suggesting that such relationships are fundamentally stable. Since SVM training is less computationally intensive and requires less data than dictionary training, SILQ implements this function on-device to allow fast adaptation to new environments.

Outage-Prediction Recall when Transferring Primitives

		SVM Training & Test Environments		
		UAV	Indoor Office	Urban Subway
Dict. Training Environment	UAV	0.81	0.82	0.74
	Indoor Office	0.83	0.79	0.79
	Urban Subway	0.81	0.81	0.73

Figure 8: Outage-prediction recall is reported when dictionary primitives are learned in one scenario and applied in another. This establishes that learned link primitives are often universal across environments and networks.

5. SILQ ARCHITECTURE

To demonstrate throughput gains from the predictions of our sparse-coding link model, we implement SILQ for Linux and Android devices. At runtime, SILQ nodes each measure their outgoing links by transmitting UDP probes at a regular interval. Reception reports are then bounced back to senders, piggybacked on probes traveling in the opposite direction. After a regular measurement interval (100ms for all results in this paper), each SILQ node computes a link prediction using the prior w measurements. If an outage is predicted, SILQ holds all data packets at the link layer until a new prediction indicates otherwise.

5.1 Measurement Protocol

SILQ's measurement protocol is shown in Figure 9. Nodes maintain a bitmap $\texttt{Link}_{in}$ marking the last w incoming probes relative to the most recent, denoted $\texttt{Seq}_{\texttt{in}}$. An arriving probe's sequence number is compared against $\texttt{Seq}_{\texttt{in}}$ to detect delivery failures before updating $\texttt{Link}_{in}$ and $\texttt{Seq}_{\texttt{in}}$. Our UDP probes use 69B to carry a reception report consisting of $\texttt{Link}_{in}$ as a 3B bitmap and $\texttt{Seq}_{\texttt{in}}$ as a 2B `unsigned short`.

Nodes also maintain a round-trip-time estimate $d_{\texttt{RTT}}$ to detect lost reception reports. For example, if $\texttt{Seq}_{\texttt{out}} = 20$

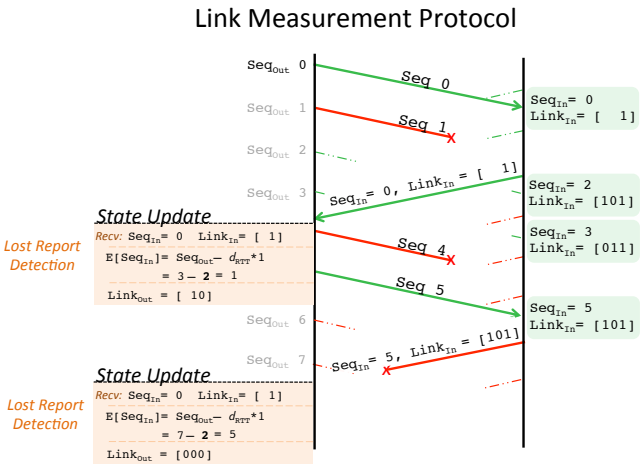

Figure 9: SILQ's measurement protocol piggybacks reception reports containing a bitmap of incoming probe receptions on outgoing probes. Senders use estimated round-trip-time d_{RTT} to detect lost reception reports.

probes are sent and only the first report successfully bounces back, the sender must compute how many reports *should* have arrived: $\texttt{Seq}_{\texttt{out}} - d_{\texttt{RTT}} \frac{w}{100\text{ms}} = 19$. This means that bi-directional probing captures asymmetric link effects, except when many reception reports are lost at the end of an interval. In this case, we make the conservative assumption that the link is off in both directions. In practice, we find that disruptions due to $d_{\texttt{RTT}}$ fluctuation resolve after one or two missed $100ms$ windows, even on a crowded carrier network.

5.1.1 *Managing Probing Overheads*

Probing consumes both power and network bandwidth, resources that must be balanced against measurement fidelity. For example, in Section 6, Figures 12 and 14 show us that sending 20 probes per 100ms consumes 22% of bandwidth for a 1Mbps link and 9% additional battery for a laptop with a power-hungry USB radio.

Beyond simply reducing probe rate, we introduce a strategy for managing overhead by updating predictions less-frequently when the link is in a stable steady-state. This *steady-state sleep* mode detects sustained strong-connectivity or sustained outage by 3 consecutive all-1 or all-0/missing link reports, respectively. In this mode, SILQ updates predictions every t_{Sleep}ms using a single round of probing. We set t_{Sleep} so that regular prediction will resume in two cases:

Case 1: Good-to-Bad Transition When a good link degrades, outgoing data will be vulnerable to loss due to a stale state prediction. We therefore compute the portion of SILQ's sending queue at risk and set t_{Sleep} to ensure tolerable loss:

$$t_{\text{Sleep}} = d_{\text{TX}} * (Q_{\text{SILQ}} - P_{\text{Min}}) - d_{\text{RTT}} \quad (3)$$

where P_{Min} is the minimum number of packets that must be delivered to maintain the connection, Q_{SILQ} is the current send-queue size, d_{TX} is the transmission time of an MTU data packet, and d_{RTT} is the time needed for a full link report to bounce back. For example, TCP uses a triple-duplicate-ACK to trigger fast recovery, so we set $P_{\text{Min}} = 3$. With $d_{\text{RTT}} = 110$ms, $d_{\text{TX}} = 12$ms, and a sending queue of $Q_{\text{SILQ}} = 50$ packets, we can set $t_{\text{sleep}} = 454$ms.

Case 2: Bad-to-Good Transition When the link is unavailable, we can save power by turning the wireless radio off and waking periodically to check link status. In this case, a large t_{sleep} risks underutilizing the link by slowing SILQ's reaction time. We tune this quantity empirically – for example we see an average 3s delay between restored connectivity and SILQ's first predicted sending opportunity in our office scenario and conservatively set $t_{\text{Sleep}} = 900$ms to ensure that the link is checked every 1s.

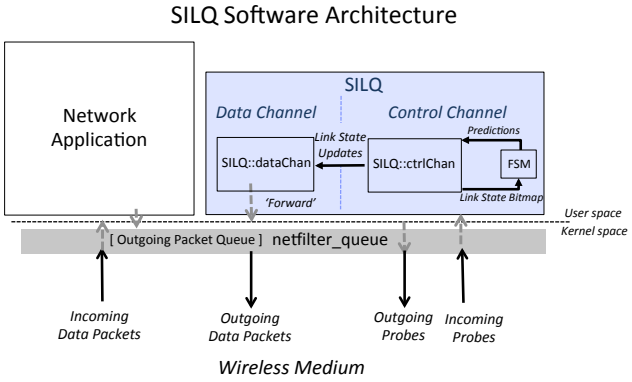

Figure 10: The SILQ architecture consists of logically separated data and control channels that isolate link measurement protocols from data forwarding protocols. In our testbed, both access the same wireless medium using the same interface.

5.2 Software Architecture & Implementation

SILQ's software architecture is shown in Figure 10. It is implemented in user space for Linux and Android devices using `netfilter_queue` to administer packets within the kernel. Each node runs both *data channel* and *control channel* controllers – the data channel controller holds or forwards packets based on the current predicted link state, while the control channel controller is responsible for probing and providing measurements to a *finite-state machine (FSM)* link-model module. The logical separation between channels simplifies probing and data-delivery protocols, however both access the same wireless medium using the same wireless interface for all results in this paper.

SILQ's FSMs are modular, allowing us to compare sparse coding to traditional modeling approaches in the field. We implement `NO_OP` (i.e. always-forward), *Loss Rate Threshold*, *Heuristic Hold*, and *Sparse Coding* FSMs. For all but the `NO_OP` FSM, trivial all-0 measurement vectors always predict outage, and trivial all-1 measurement vectors predict no-outage. For *Loss Rate Threshold* and *Heuristic Hold*, we use a "conservative" parameter to require back-to-back no-outage predictions before turning a bad link back on. This stabilization procedure is needed for loss-rate and heuristic predictors to achieve any throughput gains in the field. In contrast, we optimize the sparse coding FSM for stable prediction using the statistical methods of Section 4.2.

6. SYSTEM EVALUATION

We evaluate SILQ in the UAV, office, and subway scenarios by measuring TCP throughput, power consumption, and CPU utilization. For these experiments, we train dictionaries and SVMs as in Section 4 and then transfer a large file from a mobile device to a remote server via TCP. We compare raw TCP to SILQ using predictions from sparse coding, loss-rate threshold, and heuristic models. Our heuristic holds data packets if all probes in a previous window are lost, representing the simplest method of coping with the long out-of-range regions in our UAV and subway tests. We empirically tune a loss-rate threshold and "conservative" parameters for comparison methods to maximize throughput.

Our baseline TCP protocol is the Linux default TCP Cubic. We enable SACK-enhanced F-RTO for resilience over wireless links, use 9 keep alive probes, and allow 15 back-to-back timeouts before terminating a connection. Our wireless adapters use 7 ARQ retries for data packets, while we disable ARQ for probes over 802.11 links.

To control for differences in link quality, we compare *overall* throughput for different experiments using total data transferred divided by the total number of transmission opportunities for data packets. To illustrate the effect of small changes in experiment parameters like walking speed, Figures 11(a) and 11(b) plot loss rate for two runs in our office scenario using faint green, yellow, and red bars. We see that the durations of both outages and regions with strong connectivity differ, biasing total throughput by affording more transmission opportunities during the run shown in Figure 11(a) independent of network protocol. Our comparison metric corrects for this effect in post-processing by counting only stable regions with back-to-back probe deliveries in our throughput denominator.

6.1 UAV Throughput in Emulation

Due to logistical barriers related to personnel, budget, and weather, we validate throughput in the UAV scenario using a lightweight link emulator. During emulated flights, SILQ runs on our UAV payload and ground nodes, while probe and data packets are sent over wired interfaces to a central controller that replays field traces. The emulation controller forwards packets to their destination only if a packet reception was recorded in the field trace. We emulate a 1Mbps link and compare 1500B MTU data packets against 12 back-to-back 1ms probes to reflect a 12ms transmission time. An error in any probe causes the data packet to be dropped. The controller simulates both ARQ and CSMA. Emulated traces are held out of the training datasets used for dictionary and SVM training.

We find that SILQ boosts throughput by 22% over a raw TCP connection for links to ground-structure nodes. This improvement from 529 kbps to 644kbps is attributed to the high-quality predictions of our sparse coding link model, as heuristic and loss-rate predictions hurt throughput by 5% and 10%, respectively. For field and forest nodes, SILQ produces little benefit to average throughput, but yields an extremely low throughput deviation of 19kbps across flights.

Examining UAV links by type, we find that SILQ's sparse-coding predictions help most when link conditions vary often. Our learning approach successfully captures measurement sequences with strong predictive power caused by the complex geometry of nearby buildings and cars. In contrast, when loss is intermittent at radio range extremes, ARQ is enough to fix isolated losses. In all cases, we found that loss-rate and heuristic predictors underperform sparse coding – our heuristic only captures coarse outages lasting hundreds of milliseconds, while loss-rate predictions map both outages and intermittent losses to a single metric.

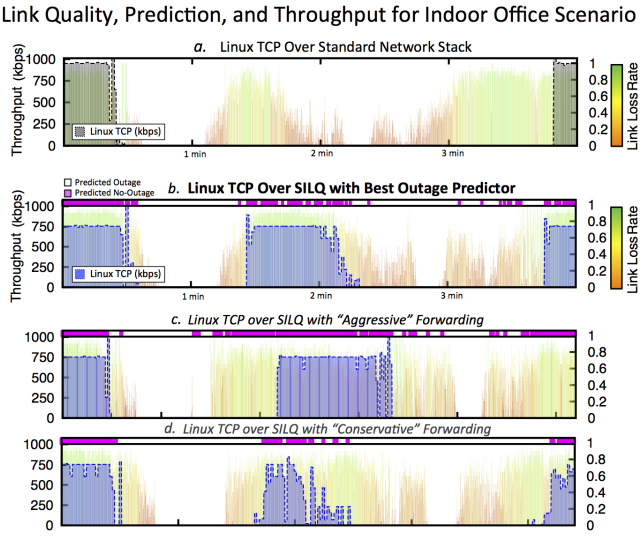

Figure 11: TCP throughput is plotted against the left axis in our indoor office scenario. For reference, link loss is plotted against the right axis using faint green, yellow, and red bars. Link-state predictions are represented by pink and white markers at the top of each plot. SILQ's optimal, aggressive, and conservative forwarders behave as anticipated, and TCP connections wake up quickly with SILQ.

6.2 Indoor Elevator/Office Environment

Next, we deploy SILQ in our indoor office environment. For these experiments, wireless links are stable when users are close to an access point, but exhibit both abrupt outages due to fire doors and gradual degradation caused by an elevator car.

Averaging five runs, SILQ improves throughput by 2x over a raw TCP connection, while reducing throughput variation by 4x. Predictions from our sparse coding model produce 15% higher throughput than the heuristic and 50% higher throughput than loss-rate predictions. We found it difficult to tune either comparison prediction-method to perform well over the entire walking tour – unlike sparse-coding, neither could cope with both abrupt link transitions *and* gradual link degradation using a single configuration. Figure 12 shows that sparse-coding predictions are also more resilient to lower probe rates than heuristic or loss-rate models. In Section 6.4, we show that this allows us to better match SILQ's power consumption to available resources.

Figure 11 shows a closer look at SILQ's performance over example runs in the indoor office. For reference, we plot link loss against the right axis using faint green and red bars. When appropriate, SILQ's link state predictions are shown at the top of a plot, with a pink marker indicating that the link is predicted to be on, and whitespace showing a predicted outage. TCP throughput is plotted against the left axis using black and blue shaded curves.

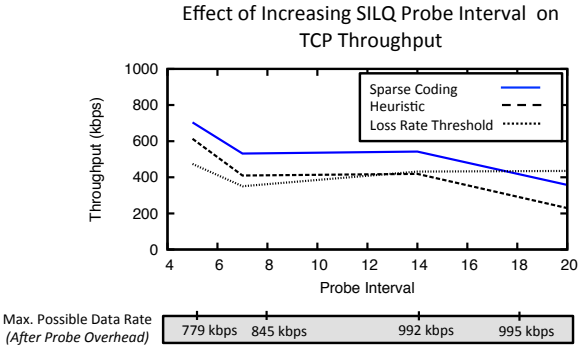

Figure 12: Average throughput in our indoor office scenario is shown as we reduce probe rate. Predictions from a sparse coding model are more resilient to low probe rates than heuristic or loss rate models.

In Figure 11(a), we see that TCP times out as soon as link conditions degrade and does not wake up until several minutes later. This causes the mobile device to miss sending opportunities during nearly 40% of the tour. In contrast, Figure 11(b) shows SILQ running with our best-tuned sparse-coding model, which predicts impending outages well and causes TCP to wake up rapidly. We also show SILQ's performance when our SVM outage predictor is tuned according to the True Positive/False Positive tradeoffs discussed in Section 4. We see how the more aggressive forwarder in Figure 11(c) sends often, but times out due to its optimism about link state. In contrast, the conservative forwarder in Figure 11(d) misses large portions of sending opportunities in an effort to avoid timeouts.

6.3 Real-World Subway Environment

Our subway scenario validates SILQ "in the wild". This scenario exhibits a great deal of uncontrolled complexity – mobile nodes experience fast changes in velocity, sudden occlusions from tunnels, spotty connectivity to 3G repeaters, and competing wireless traffic. This scenario serves to demonstrate the strength of SILQ's end-to-end design since no code alterations or network configuration changes were needed for the system to function.

Figure 13 compares raw TCP to SILQ with a sparse coding predictor over subway rides between the Harvard Square and Charles-MGH stops. Note that Figure 13(a) represents an inbound trip from Harvard to Charles-MGH, and Figure 13(b) represents an outbound trip between the two stops in the opposite order. While a raw TCP connection times out for minutes at a time, SILQ utilizes nearly every available sending window and improves throughput by 4x over a six minute train ride.

6.4 Profiling CPU & Battery Overheads

Finally, we profile SILQ's performance on two of our experimental devices: a Lenovo X200 laptop running a 1.86GHz Intel Core 2 Duo processor with a T-Link TL-WN722N USB wireless radio, and an HTC One (M8) smartphone running a 2.36GHz Qualcomm Snapdragon 801 ARM processor with onboard Wi-Fi. We run SILQ on each device with only baseline background-process activity. Throughout, screens are kept at their default brightness. On the laptop, we create network data traffic by transferring a large file via TCP to

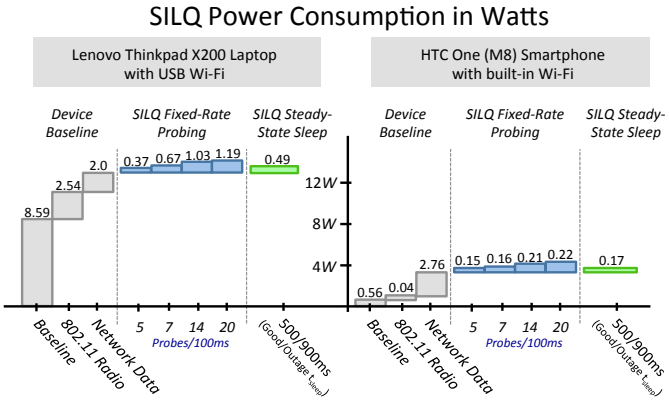

Figure 14: In our office environment, our steady-state sleep mode reduces power consumption to 4% and 5% above an active data connection, depending on device. This method forgoes predictions when the link is in a steady-state.

a remote server. On the phone, we have a user request popular web pages at a rate of roughly one site per 4 seconds. For both devices, we measure CPU utilization and battery consumption by averaging over 30 seconds.

We observe that SILQ uses only 2% CPU on the Intel chip, and 4% on the smartphone. Less than 1% of SILQ's runtime is spent computing predictions, with most time spent servicing packets. Using a dictionary of 30 atoms for $w = 20$ probes per interval, SILQ's memory footprint is 972KB.

Figure 14 shows SILQ's power consumption for each device. We see that lower probe rates reduce SILQ's power overhead significantly – for example, by 14% on the laptop and 6% on the phone when we drop from 20 to 14 probes per 100ms. We also report savings from our steady-state sleep mode in the indoor office environment. Since our devices prevent us from easily powering radios on and off without re-associating with access points, this savings is calculated in post-processing from SILQ's logs, based on recorded steady-state regions. When we compare SILQ's battery consumption overhead using the steady-state sleep mode to the power consumed by an active data connection, we see that our method only costs an additional 4% and 5% per device, utltimately for a 2-4x boost in throughput.

7. CONCLUSION

This paper presented SILQ, a link layer queuing system that uses machine learning to predict link outages so that packet loss can be avoided. Using sparse coding, we demonstrated that a link model based on combinations of simple primitives makes better use of training data and leads to higher quality predictions than models requiring steady-state assumptions about link quality. SILQ can be easily deployed on Linux and Android devices, and has been demonstrated to achieve up to 4x throughput gains in scenarios of practical interest.

Acknowledgements

This research was supported in part by gifts from the Intel Corporation. Stephen Tarsa is supported by a fellowship from the Siebel Foundation.

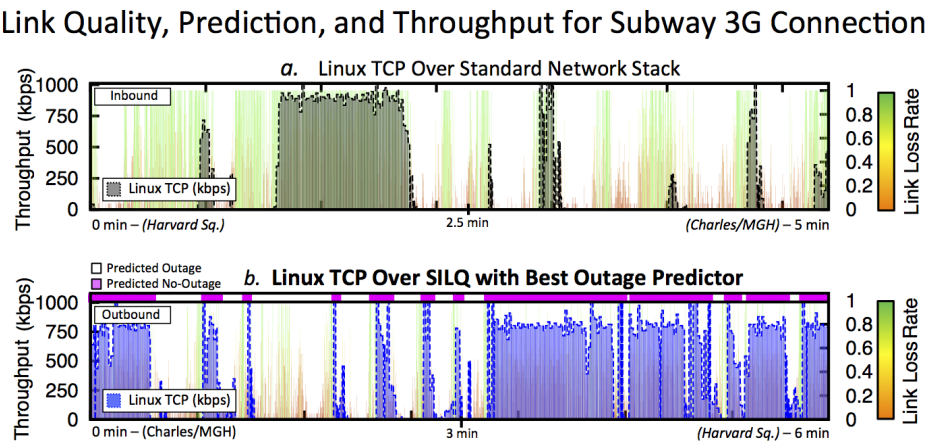

Figure 13: We compare a raw TCP connection to TCP atop SILQ using predictions from a sparse-coding link model during inbound and outbound subway rides. Over a 3G cellular network, raw TCP times out quickly and misses many sending opportunities. With SILQ, timeout is avoided and throughput increases by 4x.

8. REFERENCES

[1] J. Andrews, H. Claussen, M. Dohler, *et al.*, "Femtocells: Past, Present, and Future," *2012 IEEE Selected Areas in Communications.*

[2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *1997 IEEE/ACM Trans. on Networking.*

[3] C. Parsa and J. Garcia-Luna-Aceves, "TULIP: A Link-Level Protocol for Improving TCP over Wireless Links," in *1999 IEEE Wireless Comms. and Networking Conf.*

[4] H. Balakrishnan, S. Seshan, and R. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *1995 Wireless Networks.*

[5] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *1995 IEEE Distr. Computing Sys.*

[6] T. Goff, J. Moronski, D. Phatak, *et al.*, "Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments," in *2000 IEEE INFOCOM.*

[7] S. Tarsa and H.-T. Kung, "Hierarchical Sparse Coding for Wireless Link Prediction in an Airborne Scenario," in *2013 IEEE MILCOM.*

[8] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *1998 Springer Data Mining and Knowledge Discovery.*

[9] S. Tarsa, A. Kumar, and H.-T. Kung, "Workload Prediction for Adaptive Power Scaling Using Deep Learning," in *2014 IEEE Int. Conf. on IC Design & Tech.*

[10] B. Olshausen, D. Field, *et al.*, "Sparse Coding With An Overcomplete Basis Set: A Strategy Employed By VI?," *1997 Vision Research, Pergamon Press.*

[11] R. Valenzuela, "A Ray Tracing Approach to Predicting Indoor Wireless Transmission," in *2003 IEEE Vehicular Technology Conference.*

[12] J. Andersen *et al.*, "Propagation Measurements and Models for Wireless Communications Channels," *IEEE Comms. Magazine, 1995.*

[13] M. Ylianttila, J. Mäkelä, and K. Pahlavan, "Analysis of Handoff in a Location-Aware Vertical Multi-Access Network," *2005 Computer Networks.*

[14] E. Gilbert *et al.*, "Capacity of a Burst-Noise Channel," *1960 Bell Syst. Tech. J.*

[15] K. Farkas, T. Hossmann, F. Legendre, B. Plattner, and S. Das, "Link Quality Prediction in Mesh Networks," *2008 Computer Communications.*

[16] A. Konrad, B. Zhao, A. Joseph, and R. Ludwig, "A Markov-Based Channel Model Algorithm for Wireless Networks," *2003 Wireless Networks.*

[17] H.-T. Kung, C.-K. Lin, T.-H. Lin, S. Tarsa, and D. Vlah, "A Location-Dependent Runs-and-Gaps Model for Predicting TCP Performance over a UAV Wireless Channel," in *2010 IEEE MILCOM.*

[18] B. Ferris, D. Fox, and N. Lawrence, "WiFi-SLAM Using Gaussian Process Latent Variable Models.," in *2007 IEEE IJCAI.*

[19] T.-H. Lin and H.-T. Kung, "Stable and Efficient Representation Learning with Non-Negativity Constraints," in *2014 Intl. Conf. on Machine Learning.*

[20] K. Winstein and H. Balakrishnan, "TCP Ex-Machina: Computer-Generated Congestion Control," in *2013 ACM SIGCOMM Computer Communication Review.*

[21] H. Seferoglu, A. Markopoulou, U. Kozat, M. Civanlar, and J. Kempf, "Dynamic FEC algorithms for TFRC flows," *2010 IEEE Trans on Multimedia.*

[22] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online Learning for Matrix Factorization and Sparse Coding," *2010 Journal of Machine Learning Research.*

[23] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with Sparsity-Inducing Penalties," *2012 Foundations and Trends in Machine Learning.*

[24] R. Tibshirani, "Regression Shrinkage and Selection via the LASSO," *1996 Journal of the Royal Stats. Society.*

[25] S. Mallat and Z. Zhang, "Matching Pursuits with Time-Frequency Dictionaries," *1993 IEEE Transactions on Signal Processing.*