ePrivateEye: To the Edge and Beyond!

Chris Streiffer, Animesh Srivastava, Victor Orlikowski, Yesenia Velasco, Vincentius Martin, Nisarg Rival, Ashwin Machanavajjhala, Landon Cox

Duke University
Protecting Visual Secrets
Applications and Privacy

- Android applications take *all-or-nothing* towards system permissions
- User faced with tradeoff between usability and privacy
- Need **fine-grained** privacy control
Enter Privacy Markers

- Designed for ease of use, simple integration, and reliable performance
- User marks **public** region of their environment with privacy marker design
- Region containing **private** information will be blocked within the frame
Roadmap

- Motivation
- **PrivateEye Background**
- ePrivateEye System Design
- Evaluation
- Summary and Future Work
Android System Integration
Privacy Marker Detection Pipeline

- 5 stage pipeline consisting of 3 components
- Runtime Improvements:
  - Converting to grayscale and downsizing decrease computational complexity
  - Tracking applied every 6th frame to increase frames per second
Improving Performance

Tracking resulted in a significant increase in frame rate (~6x)
PrivateEye Limitations

- Poor performance without the use of tracking
- Tracking allows for **information leaks**, want to run full pipeline on each frame
- Does not scale to handle more frames
Goals of ePrivateEye

- Increase frame rate to 30 frames per second
  - System integration should not be noticeable at application level

- Eliminate the need for tracking
  - Improve system design so that pipeline can be run on each frame
  - Evaluations compared against PrivateEye without tracking

- Improve scalability
  - Design should be able to scale to multiple regions within frame
Roadmap

- Motivation
- PrivateEye Background
- ePrivateEye System Design
- Evaluation
- Summary and Future Work
Design Goals

- **Reduce client-side computation**
  - Migrate computationally expensive functions to the edge server
  - Client and server should send minimal amount of network traffic

- **Minimize latency between client and server**
  - Control aspects of communication to ensure minimal delay
  - Integrate Software Defined Networking to ensure fast-path for communication
Reduce Client-side Computation

- Divide the computations between the client and server
- Client responsible for:
  - Converting image to grayscale
  - Reducing image size
  - Applying occlusion filter to original image
- Server responsible for:
  - Running edge detection

![Diagram showing the process flow between client and server](image-url)
Minimize Latency

- Communication across TCP
  - Maintain persistent TCP connections
  - Disable Nagle’s algorithm on the client side
  - Reduce number of `read` syscalls by modifying the size of the input buffer

- Software Defined Networking Configuration
  - Established a mostly fiber path between the client and remote server
  - Allows for communication path to bypass firewalls
Evaluation Configurations

- **Baseline Configurations**
  - PrivateEye running on-device with tracking **disabled**

- **Managing Network Conditions**
  - Edge
  - Building
  - Metro

- **Case Studies**
  - Degraded network conditions
  - Cloud implementation
Edge Configuration

- 1.3 GHz Intel Core M CPU and 8 GB of RAM
- Transmission performed using 802.11ac
- Access point in same room as device
Building Configuration

- 1.3 GHz Intel Core M CPU and 8 GB of RAM
- Transmission performed using 802.11n
- Device and access point connected to same wireless router
Metro Configuration

- 6 Intel Xeon Processor 2.10 GHz v4 CPUs with 128 GB of RAM, 1 Tesla P100 GPU
- Access point and server connected across a primarily fiber-optic connection
- Device and server located approximately 2.85 miles apart
Roadmap

- Motivation
- PrivateEye Background
- ePrivateEye System Design
- Evaluation
- Summary and Future Work
Performance Evaluation

- Video benchmark statistics:
  - 20s long videos with resolution 1280x960
  - Shot in 3 different video configurations

- Video action configurations:
  - Still – simulates image capture
  - Spin – to understand the impact of change in orientation
  - Scan – simulates a video recording

- Record and Replay
  - Initially record videos described above
  - Replay captured videos to maintain consistency between evaluations
Frames per Second Results

[Bar chart showing frames per second for different video configurations (Still, Spin, Scan) across different categories (Local, Edge, Building, Metro).]
Scalability Evaluation

- **Video benchmark test suite:**
  - 20s long videos with resolution 1280x960
  - Each video configured using **still** action

- **Video configurations:**
  - Contain 1, 2, and 3 marked regions
  - Increase in regions corresponds to an increase in CPU load
Scalability Results

![Graph showing scalability results with bars representing frames per second for different numbers of marked regions. The graph compares 'Edge' and 'Local' performance.]
Case Study: Degraded Network

- Use the **netem** Linux package to induce packet loss and network latency
- Slow path
  - Add external latency on the server side
  - Increment latency from 0-70ms in increments of 10ms
- Lossy path
  - Randomly inject packet loss into the network on the server end
  - Increment the percentage of packet loss from 0-10% in increments of 1%
Degraded Network Configuration

- Loss and latency injected into the network on the server end
- Communication between access point and server uses **SDN fast-path**
- Only loss within network from **netem** modifications
Slow Path Results

System design was able to sustain high performance with latency up to 30ms
Lossy Path Results
Case Study: Cloud

- **AWS Server Configuration**
  - Located in **us-east-1** availability zone
  - Average ping time of 19.275ms
  - Packet loss of 2.3%

- **Client and Server Communication**
  - All data encrypted using a **ssh tunnel**
  - Presents ideal scenario for running in a cloud environment
AWS Configuration

- Communication maintained between access point and AWS server using SSH tunnel
- All public data encrypted once it leaves the trusted network
AWS Results
Summary

● Offloading Impact
  ○ Utilize edge infrastructure to improve the quality of service
  ○ Achieved 30 fps, eliminated tracking, scaled across multiple markers

● Managing Network Conditions
  ○ Determined empirical bounds for running edge configuration
  ○ Edge against cloud configuration

● Increasing Computational Complexity
  ○ Metro configuration allows streamlined communication to high-performance GPU
  ○ Opens door for deep learning algorithms which require more resources