Secure Control Interface for a PC Using a Mobile Device

Wesley Minner
CS 217B, Spring 2017
Project Final Presentation

Motivation (Recap)

Primary use case: **slideshow control**

- Traditional...
  - Wired: short cord → small radius
  - Wireless: poor security, possibly spotty signal, dead batteries
  - Must carry around extra hardware
- ndnMouse…
  - Robust security
  - Efficient multicasting
  - Uses pre-existing hardware (your phone) and wireless resources (a local WiFi access point or phone hotspot)

Features

- Full mouse control
  - Relative cursor movement
  - Left/right click (with hold down support)
  - Movement sensitivity and precision settings
- Two-finger scrolling
  - Works similarly to Apple laptops
  - Scrolling inversion and sensitivity settings
- Rudimentary keyboard support
  - Common slideshow control commands: arrows keys, spacebar, ESC, etc…
  - Custom typed messages: using built-in Android keyboard
- Encompassing security
  - Defends against packet snooping, replay attacks, privacy attacks, and brute force attacks

Supported Platforms

ndnMouse composed of two applications...

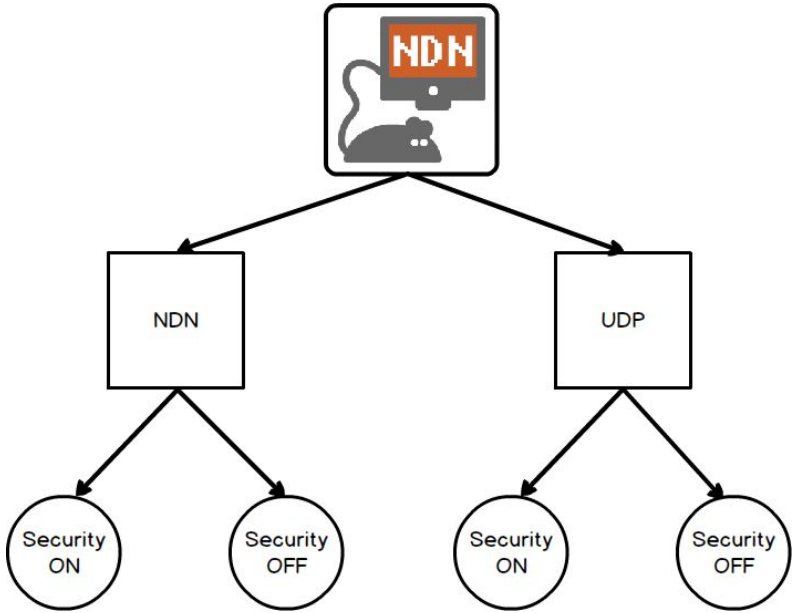
- Server/producer Java application
  - Running on any relatively modern Android phone (Android 4.1 and up)
- Client/consumer Python application
  - Running on any PC that can run NFD and Python3
  - Windows still supported by ndnMouse's UDP communication

Communication Protocols

Four Protocol Configurations

- Communication over…
  - UDP
  - NDN
- Security…
  - ON
  - OFF

2 comm choices x 2 security choices = 4

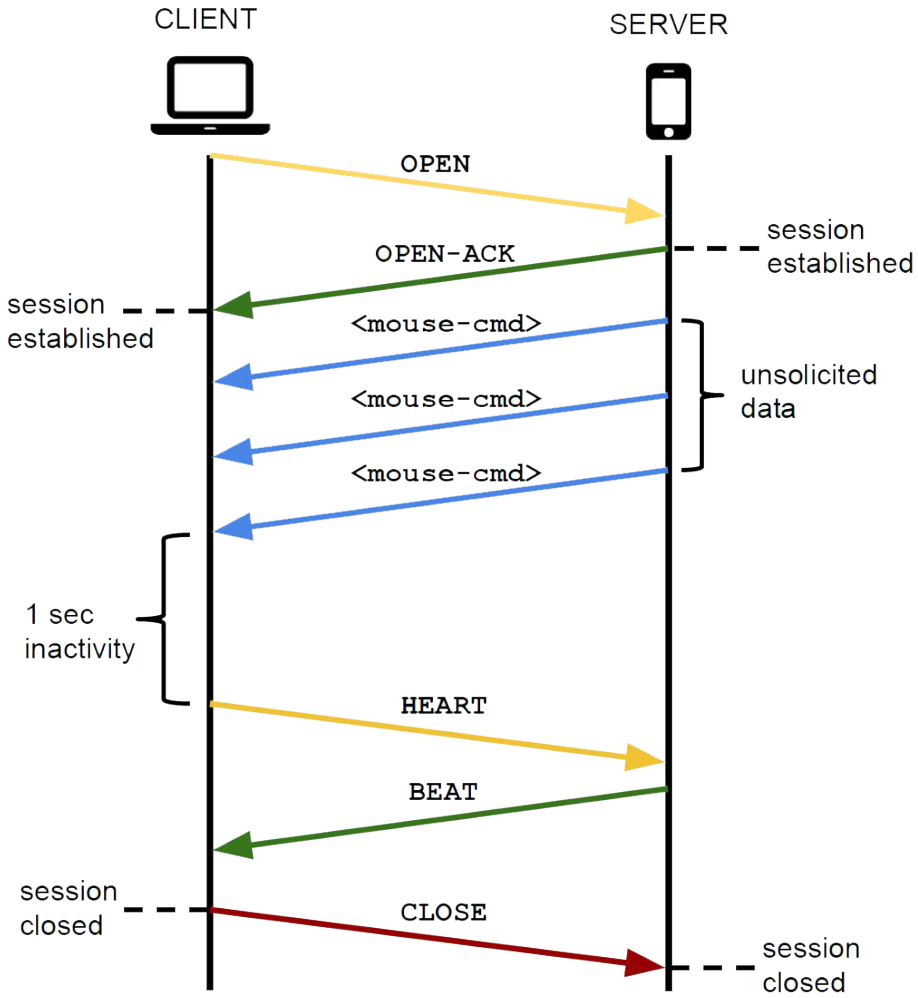


Protocol Requirements

- ndnMouse **DOESN'T** need:
  - Reliable data delivery
  - In-order processing
- ndnMouse **DOES** need:
  - Recovery from network loss
  - Low latency
- Late packets will be thrown away to prevent unexpected mouse movement and jitter

UDP

- ndnMouse's baseline, IP-based protocol
  - High performance
  - Unsolicited data
- UDP satisfies the requirements better than TCP
- Stateful session implemented on top of UDP



UDP Message Formats

Message Type	Message Format
Movement	M<x-4B><y-4B>
Click	C<click_message>
Scroll	S<x-4B><y-4B>
Keyboard	K<keyboard_message>
Typestring	T<type_string-10B-max>

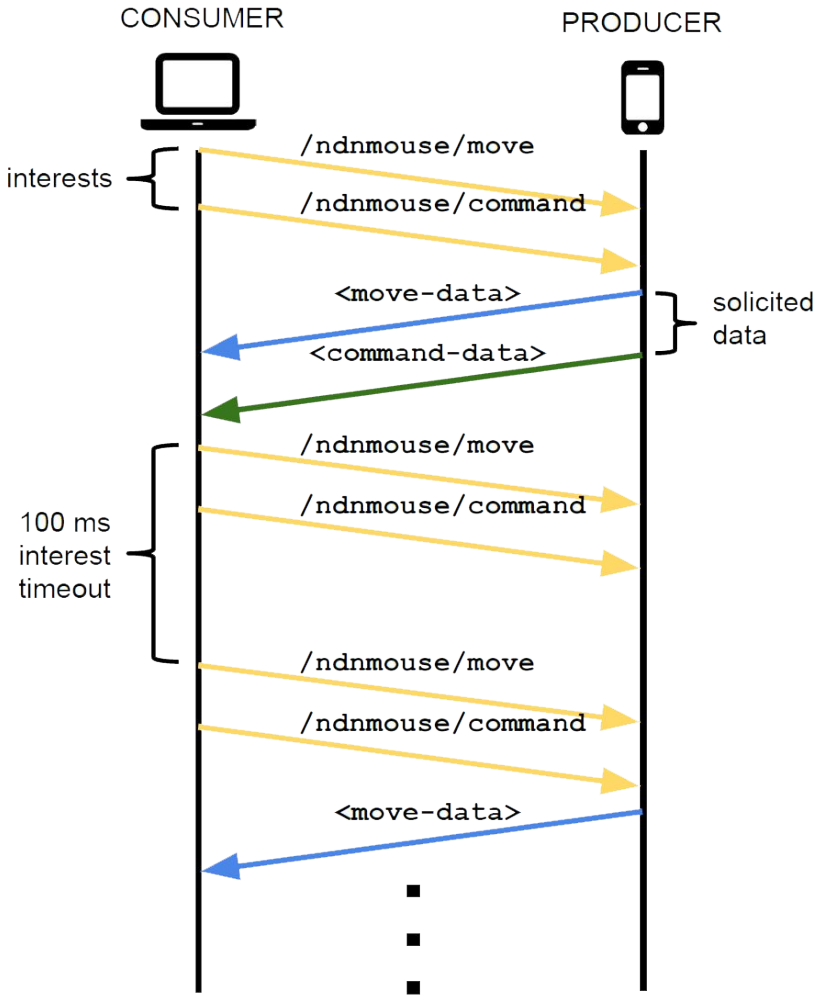
Table 1: Non-secure message formats

Message Type	Secure Message Format
Movement	<seq-4B>M<x-4B><y-4B>
Click	<seq-4B>C<click_message>
Scroll	<seq-4B>S<x-4B><y-4B>
Keyboard	<seq-4B>K<keyboard_message>
Typestring	<seq-4B>T<type_string-10B-max>

Table 2: Secure message formats

NDN

- ndnMouse's primary protocol
  - Simple
  - Stateless
  - Solicited data
- No unsolicited data
  - One interest → one data
- May be multiple interests pending at any one time



NDN Message Formats

- Mouse control interests
  - /move
  - /command
- Security related interests
  - /seq
  - /salt

Interest Name	Purpose
`/ndnmouse/move`	Movement Data
`/ndnmouse/command`	Command Data
`/ndnmouse/seq`	Sequence Number Sync
`/ndnmouse/salt`	Password Salt Data

Table 3: NDN served interests

Mouse Packets

- Fixed length 32 byte packets
- Cleartext
  - Random 16 byte initialization vector (IV)
- Ciphertext
  - Sequence number, 4 bytes
  - Message padded to 12 bytes with PKCS5 padding

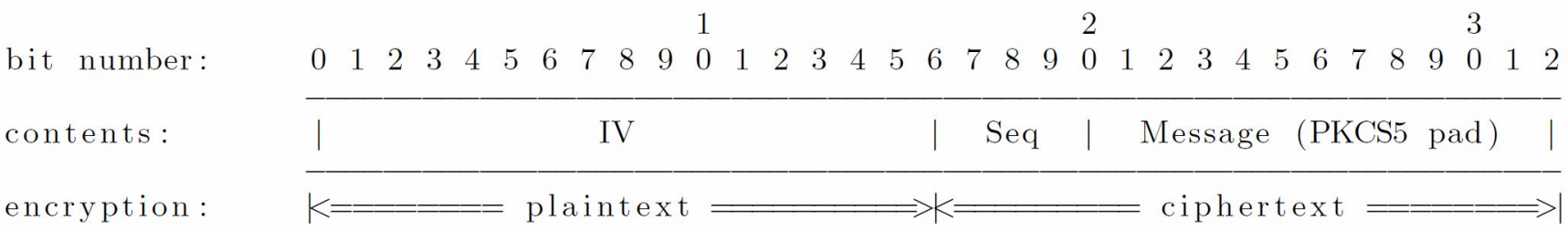


Figure 1: Secure mouse packet description

Security

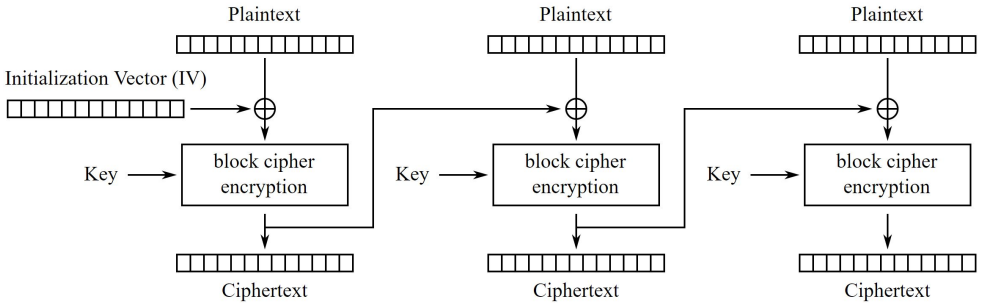
Three Major Components

- Data Encryption*
- Sequence Number Validation
- Password Salting

* We get **User Authentication** for free from data encryption

Data Encryption

- Advanced Encryption Standard (AES)
  - Using 128 bit key hashed from user password + salt (SHA256)
  - Only consumers with proper key can decrypt → authentication
- Cipher block chaining (CBC)
  - 16 byte block size
- Random IVs for each packet
  - Prepended to encrypted payload
  - Communicated in cleartext



Cipher Block Chaining (CBC) mode encryption

Sequence Number Validation

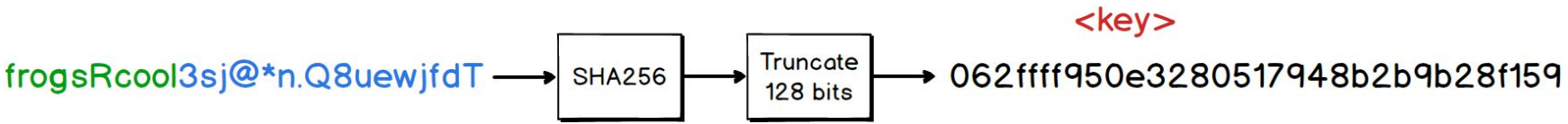
- Protects against ***inter*-session** replay attacks
- Enforced policy: *no command should be executed which contains a sequence number lower than the largest sequence number witnessed by the device*
  - Policy applies to both UDP and NDN, clients/consumers and server/producer
- UDP
  - **`OPEN`** message carries seq num 0
  - **`OPEN-ACK`** response carries seq num 1
- NDN
  - Interests do not carry seq num (security reasons)
  - Response data carries seq num only
- Catch-up mechanisms help out of sync devices recover
  - /ndnmouse/seq

Password Salting

- Protects against ***intra*-session** replay attacks
- UDP
  - Uses IV of initial **`OPEN`** message as salt
  - Each client gets a different salt per session
- NDN
  - Consumer requests salt directly from producer: /ndnmouse/salt
  - Each consumer gets the same salt per session

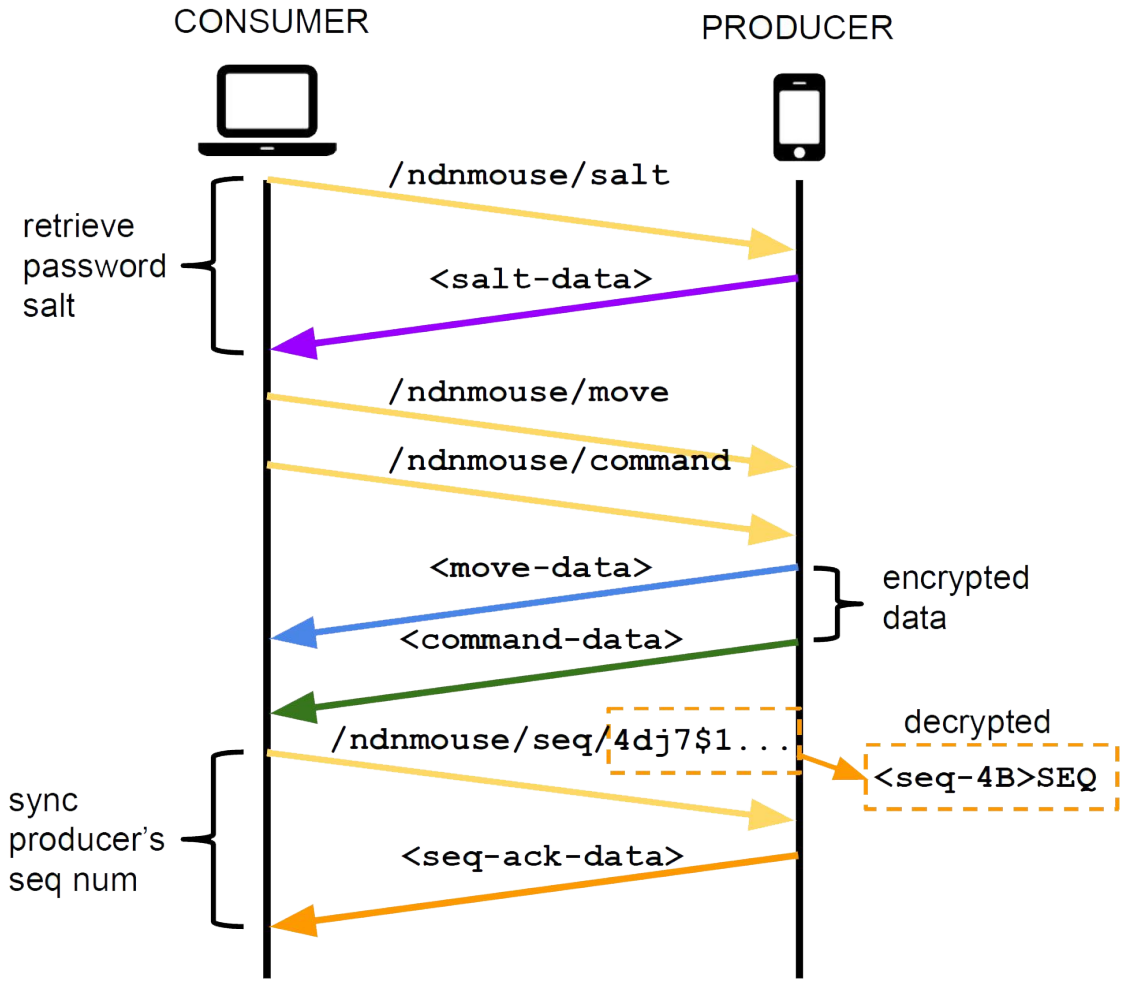
<password> <salt>

frogsRcool + 3sj@*n.Q8uewjfdT = frogsRcool3sj@*n.Q8uewjfdT



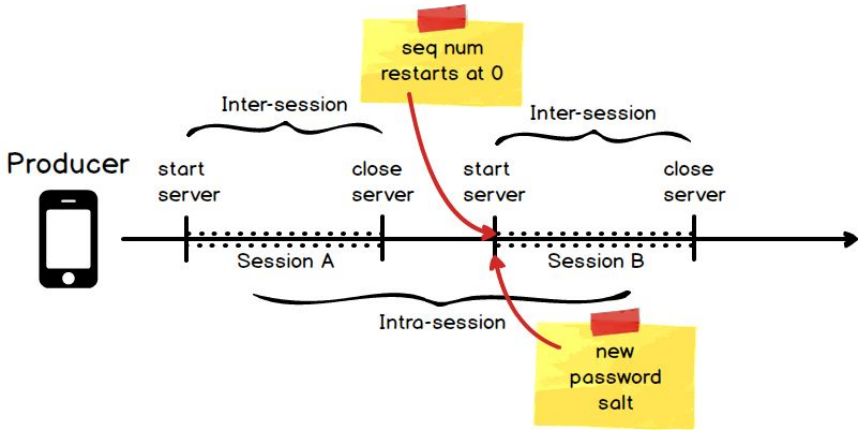
Security Interests

- Password salt
  - unique for each producer session
- Seq num sync
  - protected by checking decrypted message format



Attack Types and Defenses

- Snooping data
  - Encrypting payloads
- Replay attacks
  - Sequence number validation (inter-session)
  - Password salts (intra-session)
- Privacy attacks
  - Random IVs on each packet
  - (Two packets with same payload encrypt to different ciphertext)
- Brute force attacks
  - Short-lived, unique keys per session via password salt



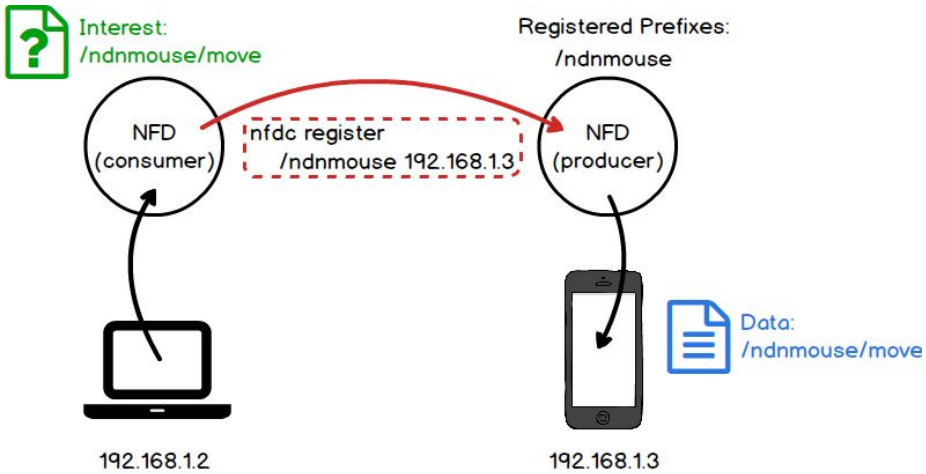
Challenges and Trade-offs

Unsolicited Data

- Background
  - Unsolicited data useful for sending **unpredictable** mouse command data like mouse clicks, keyboard presses, etc…
  - Client can avoid the need to poll for this type of data
  - UDP can easily send unsolicited data
- Problem
  - NDN cannot send unsolicited data (one interest → one data)
- Solution
  - Create additional interests to poll continuously: /ndnmouse/command
- Trade-off
  - **Performance** hit from additional outstanding interests
  - **Complexity** of packing/unpacking data (get all data from one big interest)

Addressing Devices Using NDN

- Problem
  - NFD does not currently propagate prefix registrations
  - ndnMouse uses a common WiFi AP → two NFD hops between consumer and producer
- Example
  - Producer registers prefix /ndnmouse with its local NFD
  - Registration does not propagate to other NFDs
  - Consumer NFD doesn't know where to forward /ndnmouse interests
- Solution
  - ndnMouse asks consumer to enter IP address of producer
  - Set up a route for NFD to forward interests



Signature Validation vs Shared Password

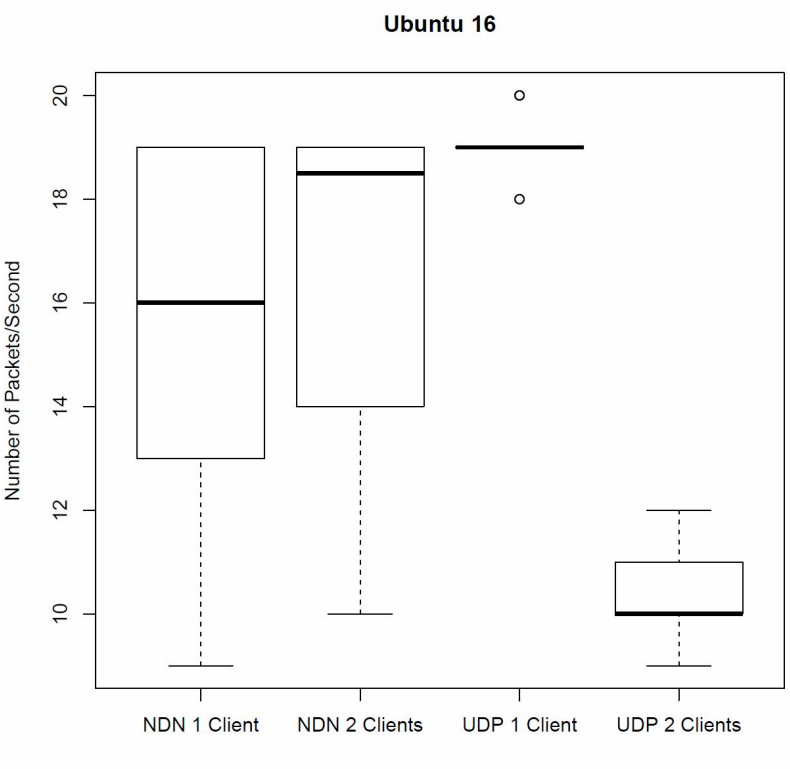
- Problem
  - Need to share a secret (symmetric key) between consumer and producer
  - How does producer know which consumers to trust?
- Solution
  - User must whitelist certain device identities (requires signed identity installation ahead of time)
  - Then validate devices by traveling up trust chain to trust anchor
  - Then use public/private keys to exchange symmetric key
- OR more practical solution...
  - User could provide a password on both devices, since they are starting up the application anyway
- Trade-off
  - **Complexity** of requiring user to have intimate knowledge of NDN (signing/whitelisting identities)
  - **Long-term overhead** of requiring a password on every startup

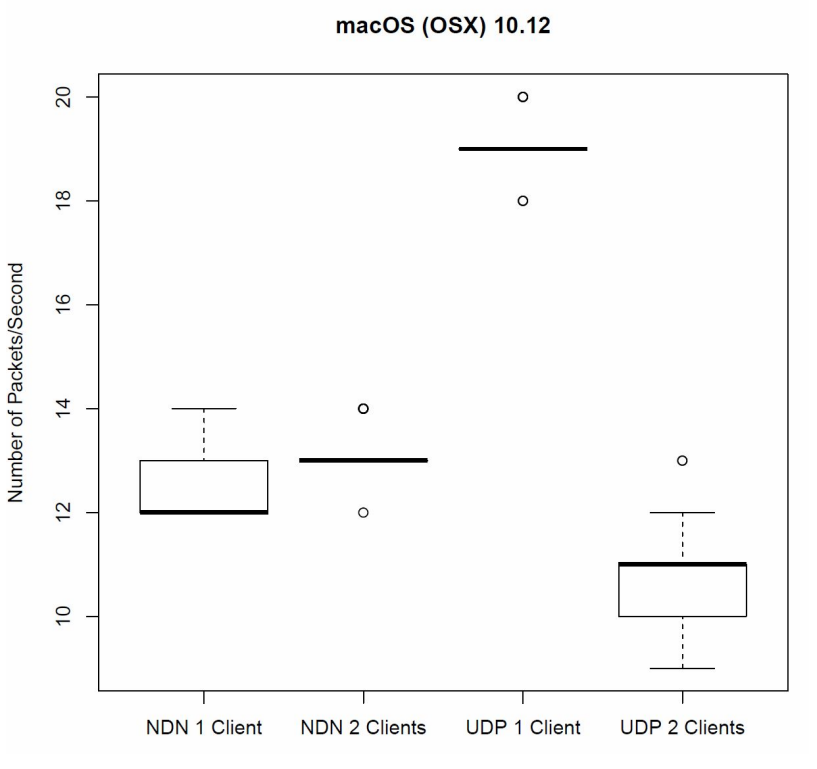
Performance Analysis

Benchmark Setup

- Devices
  - Custom built desktop: Ivy Bridge i5 Intel CPU, Ubuntu 16
  - Macbook Pro: late 2013 model, macOS Sierra 10.12
  - Nexus 5X Android phone: Android 7.0
- All devices on same wireless WiFi access point
- NdnMouse security enabled for all benchmarks
- Each test lasted 30 seconds, sending continuous movement updates
  - Ideal (maximum) movement update frequency: 20 packets/sec

NDN vs UDP


Ubuntu 16
Number of Packets/Second
10
12
14
16
18
20
NDN 1 Client
NDN 2 Clients
UDP 1 Client
UDP 2 Clients


macOS (OSX) 10.12
Number of Packets/Second
10
12
14
16
18
20
NDN 1 Client
NDN 2 Clients
UDP 1 Client
UDP 2 Clients

Event-Driven Architecture (NDN)

- Program flow
  - A single thread is created on the phone to run the NDN producer
  - Producer sets up callbacks for specific interest names (events)
  - NFD condenses duplicate interests and forwards unique ones
  - Interests arrive and producer handles them serially
- Advantages
  - Efficient and scales well
  - Encourages stateless design → shorter and simpler code
- Disadvantages
  - Performance limited by NFD overhead
  - Hard to add per-client state if needed

Multi-Threaded Architecture (UDP)

- Program flow
  - One parent thread is created on the phone to run the server
  - Server spins up a separate worker thread to handle each new client
  - Worker thread dies only when client ends session
- Advantages
  - Easy to keep per-client state
  - UDP integrated in kernel gives very fast performance
- Disadvantages
  - Multiple clients increase thread management overhead → degraded performance
  - Multicasting not efficient

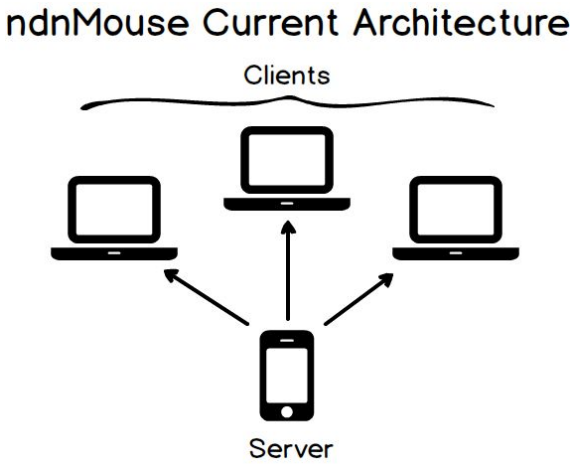
Extensions

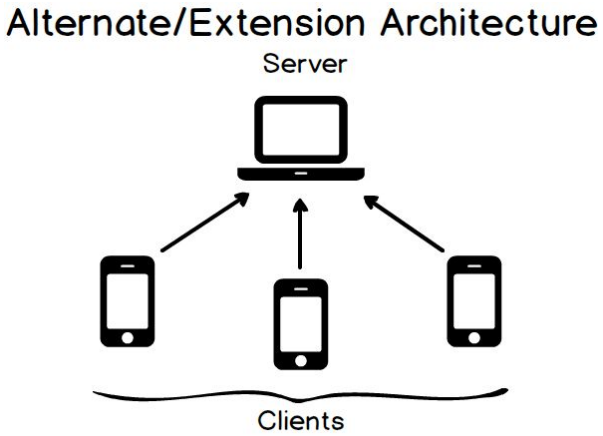
NDN Performance Optimization

- Multiple outstanding interests
  - Problem: must wait for the single /ndnmouse/move interest to come back (data or timeout) before sending out the next movement update interest
  - Possible solution: rotate between /ndnmouse/move1, /ndnmouse/move2, /ndnmouse/move3
    - Alternate solution: append sequence numbers → /ndnmouse/move/<seq>
  - Trade-off: client latency with server memory/processing
- Condense interests into one big interest
  - Problem: currently separate interests for different types of data
    - /ndnmouse/move
    - /ndnmouse/command
  - Possible solution: merge interests into a generalized data interest
    - /ndnmouse/move + /ndnmouse/command → /ndnmouse/update
  - Trade-off: client latency with data packing/unpacking complexity

Many Mice, One PC

- Current: control multiple PCs with one mouse
  - Ex: roadmap to cross-computer mouse support (see two slides from now)
- Extension: control one PC with many mice
  - Ex: shared slideshow control





Additional Wireless Interface Support

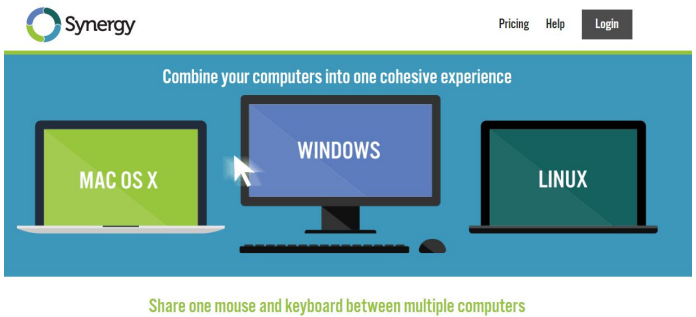
- Users may wish to use wireless interfaces other than WiFi AP
- Bluetooth
  - Being worked on by Da Teng
- WiFi Direct
  - Being worked on by Amar Chandole

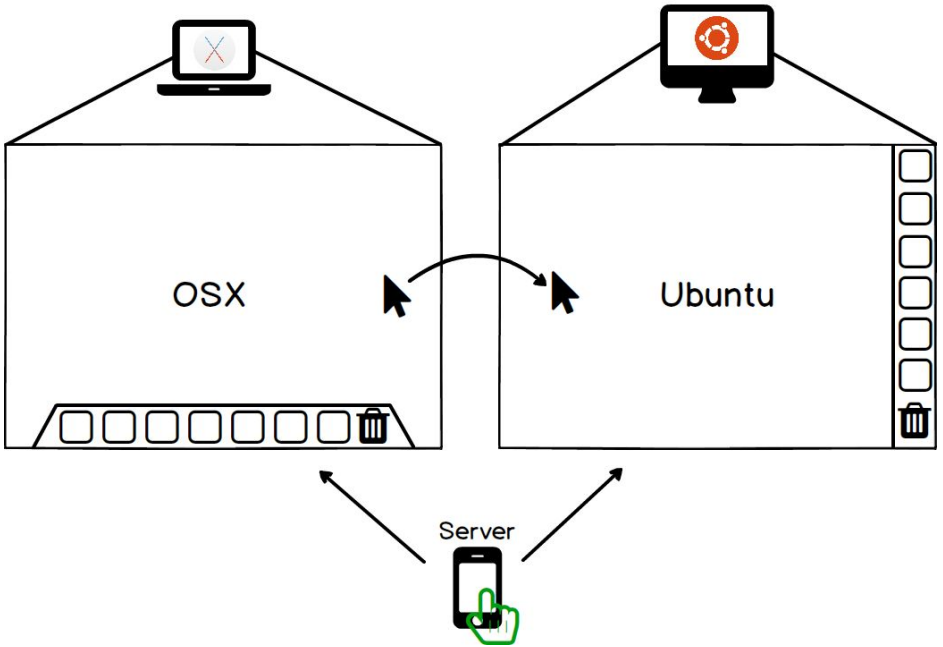
From our collaborations, minor changes needed for ndnMouse to support these interfaces.

- Additional questions: is security necessary on bluetooth?

Cross-Computer Mouse Support

- Many computers/OS's can share a mouse/keyboard
- Synergy already does this for PCs
- NdnMouse could do this using your phone, with more efficient multicasting!

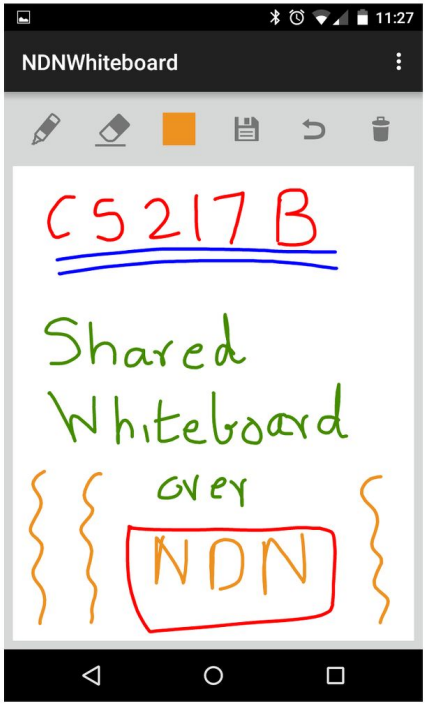




Related Work

NDN Applications

- No known NDN real-time control applications
- ChronoSync offers real-time synchronization of data
    - Applications that use this, like NDN Whiteboard, are more latency tolerant than ndnMouse



Open-Source Mouse Control

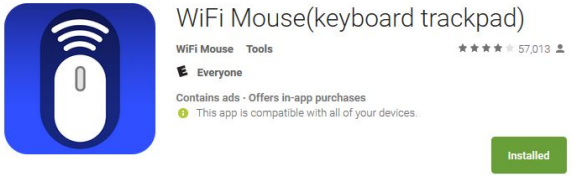
- Android Desktop Remote Control (53 stars on Github)
  - UDP datagrams
  - Server-PC/client-phone: however, does not support many-mice/one-PC relationship
  - No security
  - Not released on Google Play
- Mobile Mouse (8 stars on Github)
  - TCP socket
  - Server-PC/client-phone: however, does not support many-mice/one-PC relationship
  - No security
  - Not released on Google Play

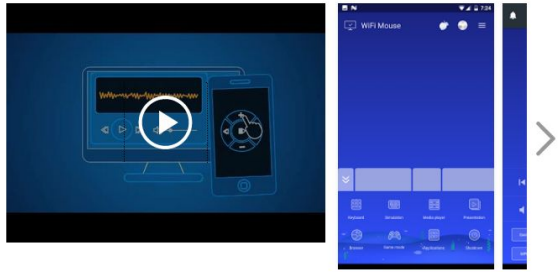
Closed-Source Mouse Control

- Remote Mouse (5-10 million installs)
  - Free base application: ad-supported + additional pay features
  - Server-PC/client-phone: supports many-mice/one-PC relationship
  - UDP for commands and TCP for session setup
  - Recovers from temporary network loss
  - Very weak security
    - No encryption
    - Same security hash used every time for authorization on session setup (easy to replay)
    - Mouse movements in cleartext
    - Keyboard typing obfuscated by some form of a Ceasar Cipher (shifting characters by a constant amount)

Closed-Source Mouse Control

- WiFi Mouse (5-10 million installs)
  - Free base application: ad-supported + additional pay features
  - Server-PC/client-phone: supports many-mice/one-PC relationship
  - TCP for all communication
  - Does NOT recover from temporary network loss
  - Extremely Weak Security
    - No encryption
    - Same security hash used every time for authorization on session setup (easy to replay)
    - Mouse and keyboard commands all in cleartext
  - Privacy concerns: transmits what programs you are running on your PC to your phone





Wireshark · Follow TCP Stream (tcp.stream eq 12) · wireshark_1C

```
system windows 6.2
needpassword
verifypassword 5089294a6560d748b5d8fcb14f7f4bec
verifypassword ok
reportCurrentApp
currentapp wireshark.exe
currentapp searchui.exe
currentapp explorer.exe
currentapp snippingtool.exe
currentapp evernote.exe
dontreportCurrentApp
utf8 p
utf8 a
utf8 s
utf8 s
utf8 w
utf8 o
utf8 r
utf8 d
```

What Have I Learned?

- Don’t use closed-source mouse control applications
- Adding strong security to an insecure implementation is difficult
  - Not easy to just “patch” it in. Many changes to the communication protocol had to be made.
  - Sometimes statefulness is a necessary evil (to prevent replay attacks!)
- NDN applications require a new way of thinking about data
  - How can we reuse data?
  - How do we exploit caches (or avoid them altogether)?

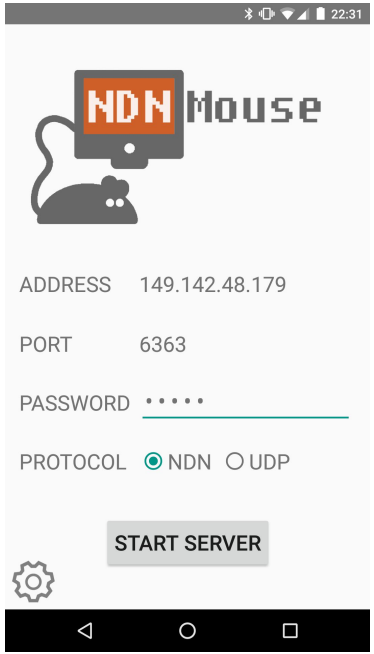
In NDN, multicast is the star of the show!

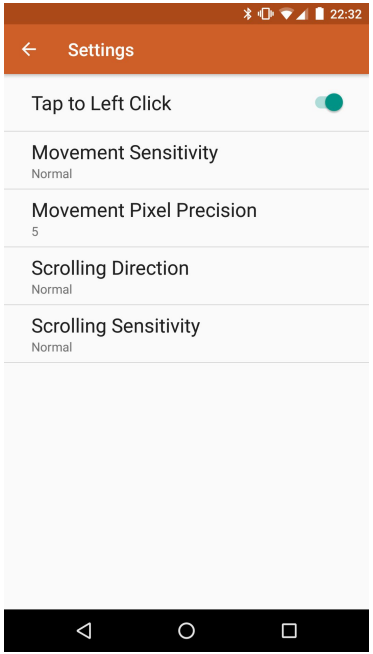
Thank You!

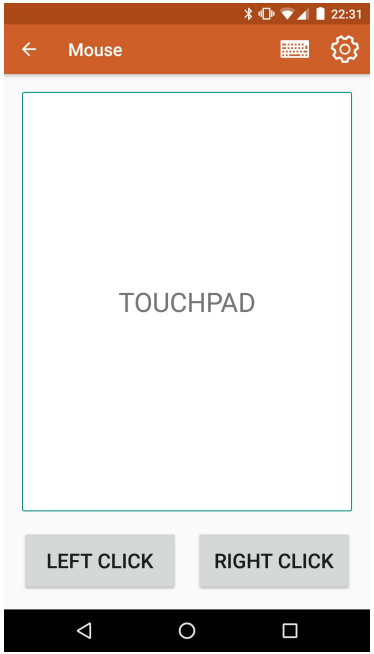
References

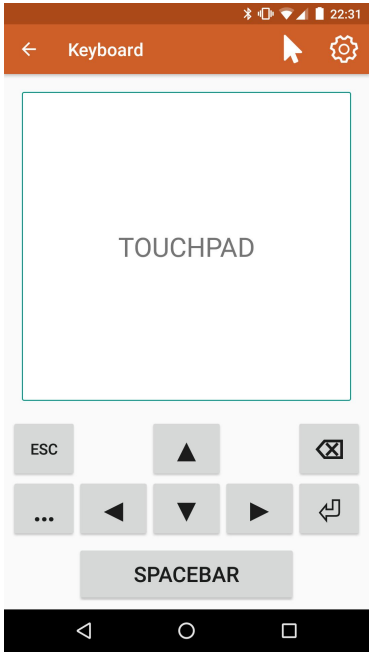
- CBC: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
- Android Desktop Remote Control: https://github.com/justin-taylor/Android-Desktop-Remote-Control
- Mobile Mouse: https://github.com/tuesda/mobilemouse
- Remote Mouse: https://play.google.com/store/apps/details?id=com.hungrybolo.remotemouseandroid
- WiFi Mouse: https://play.google.com/store/apps/details?id=com.necta.wifimousefree

Appendix - Screenshots









Appendix - Screenshots