Lecture 11: VLAN, MPLS, Network Security

VLANs: motivation

- CS user moves office to EE, but wants connect to CS switch?
- Single broadcast domain:
  - All layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - Security/privacy, efficiency issues

VLANs

- Port-based VLAN: switch ports grouped (by switch management software) so that single physical switch ……

- Traffic isolation: frames to/from ports 1-8 can only reach ports 1-8
  - Can also define VLAN based on MAC addresses of endpoints, rather than switch port

- Dynamic membership: ports can be dynamically assigned among VLANs

- Forwarding between VLANs: done via routing (just as with separate switches)
  - In practice vendors sell combined switches plus routers

VLANs spanning multiple switches

- Trunk port: carries frames between VLANs defined over multiple physical switches
  - Frames forwarded within VLAN between switches can’t be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1Q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN frame format
MPLS and IP router

- **IP routing**: path to destination determined by destination address alone
- **MPLS routing**: path to destination can be based on source and dest. address
  - **fast reroute**: precompute backup routes in case of link failure

MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing.
  - e.g., link bandwidth, amount of “reserved” link bandwidth
- entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers

MPLS forwarding tables
Data center networks

- 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
  - e-business (e.g., Amazon)
  - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
  - search engines, data mining (e.g., Google)

- challenges:
  - multiple applications, each serving massive numbers of clients
  - managing/balancing load, avoiding processing, networking, data bottlenecks

---

Data center networks

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy

---

Synthesis: a day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - goal: identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - scenario: student attaches laptop to campus network, requests/receives www.google.com

---

Link layer, LANs: outline

5.1 introduction, services
5.2 error detection, correction
5.3 multiple access protocols
5.4 LANs
  - addressing, ARP
  - Ethernet
  - switches
  - VLANS
5.5 link virtualization: MPLS
5.6 data center networking
5.7 a day in the life of a web request

---

A day in the life: scenario
A day in the life… connecting to the Internet

- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFDFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

A day in the life… ARP (before DNS, before HTTP)

- before sending HTTP request, need IP address of www.google.com: DNS
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

A day in the life… TCP connection carrying HTTP

- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!

Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life… using DNS

- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

web page finally (!!!) displayed

- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client
Chapter 8: Network Security

**Chapter goals:**
- understand principles of network security:
  - cryptography and its many uses beyond "confidentiality"
  - authentication
  - message integrity
- security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers

What is network security?

**Confidentiality:** only sender, intended receiver should "understand" message contents
- sender encrypts message
- receiver decrypts message

**Authentication:** sender, receiver want to confirm identity of each other

**Message integrity:** sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

**Access and availability:** services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages

Who might Bob, Alice be?

- … well, real-life Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

There are bad guys (and girls) out there!

Q: What can a “bad guy” do?
A: A lot! See section 1.6
- *eavesdrop*: intercept messages
- actively *insert* messages into connection
- *impersonation*: can fake (spoof) source address in packet (or any field in packet)
- *hijacking*: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

The language of cryptography

<table>
<thead>
<tr>
<th>plaintext</th>
<th>encryption algorithm</th>
<th>ciphertext</th>
<th>decryption algorithm</th>
<th>plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice’s encryption key</td>
<td>$K_A$</td>
<td>ciphertext, encrypted with key $K_A$</td>
<td>$m = K_A(m)$</td>
<td>plaintext</td>
</tr>
<tr>
<td>Bob’s encryption key</td>
<td>$K_B$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

m | plaintext message
$K_A(m)$ | ciphertext, encrypted with key $K_A$
| $m = K_A(m)$ | plaintext|
Breaking an encryption scheme

- **cipher-text only attack**: Trudy has ciphertext she can analyze
- **known-plaintext attack**: Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a, i, c, e, b, o.
- **chosen-plaintext attack**: Trudy can get ciphertext for chosen plaintext

Simple encryption scheme

**substitution cipher**: substituting one thing for another
- monoalphabetic cipher: substitute one letter for another

plaintext:  abcdefghijklmnopqrstuvwxyz
ciphertext:  mnbvcxzasdfghjklpoiuytrewq

e.g.: Plaintext: bob. i love you. alice
ciphertext:  nkn. s gktc wky. mgsbc

**Encryption key**: mapping from set of 26 letters to set of 26 letters

Symmetric key crypto: DES

**DES: Data Encryption Standard**
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- **how secure is DES?**
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- **making DES more secure**
  - 3DES: encrypt 3 times with 3 different keys

Symmetric key crypto: DES

**DES operation**
initial permutation
16 identical “rounds” of function application, each using different 48 bits of key
final permutation
**AES: Advanced Encryption Standard**

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

**Public Key Cryptography**

**symmetric key crypto**

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

**public key crypto**

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do not share secret key
- public encryption key known to all
- private decryption key known only to receiver

**Public key cryptography**

**plaintext** message, \( m \)

**encryption** algorithm \( K_B(m) \)

**ciphertext**

**decryption** algorithm \( K_B^{-1}(m) \)

**plaintext** message

**Bob's public key** \( K_B^* \)

**Bob's private key** \( K_B^{-1} \)

**requirements:**

1. need \( K_B^*(\cdot) \) and \( K_B^{-1}(\cdot) \) such that
   \[
   K_B^{-1}(K_B^*(m)) = m
   \]
2. given public key \( K_B^* \), it should be impossible to compute private key \( K_B^{-1} \)

**RSA:** Rivest, Shamir, Adelson algorithm

**Prerequisite: modular arithmetic**

- \( x \mod n = \text{remainder of } x \text{ when divide by } n \)
- facts:
  - \([a \mod n] + (b \mod n) \mod n = (a+b) \mod n\)
  - \([a \mod n] - (b \mod n) \mod n = (a-b) \mod n\)
  - \([a \mod n] \ast (b \mod n) \mod n = (a\ast b) \mod n\)
- thus
  - \((a \mod n)^d \mod n = a^d \mod n\)
- example: \( x=14, n=10, d=2\):
  - \((x \mod n)^d \mod n = a^d \mod 10 = 6\)
  - \(x^2 = 14^2 = 196 \mod 10 = 6\)

**RSA: getting ready**

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number.

**example:**

- \( m = 10010001 \). This message is uniquely represented by the decimal number 145.
- to encrypt \( m \), we encrypt the corresponding number, which gives a new number (the ciphertext).
RSA: Creating public/private key pair

1. choose two large prime numbers \( p, q \)
   (e.g., 1024 bits each)
2. compute \( n = pq \), \( z = (p-1)(q-1) \)
3. choose \( e \) (with \( e < n \)) that has no common factors with \( z \) (\( e, z \) are "relatively prime").
4. choose \( d \) such that \( ed - 1 \) is exactly divisible by \( z \).
   (in other words: \( ed \mod z = 1 \)).
5. public key is \((n,e)\). private key is \((n,d)\).

RSA: encryption, decryption

0. given \((n,e)\) and \((n,d)\) as computed above
1. to encrypt message \( m (<n) \), compute
   \[ c = m^e \mod n \]
2. to decrypt received bit pattern, \( c \), compute
   \[ m = c^d \mod n \]

Why does RSA work?

- must show that \( c^d \mod n = m \)
- fact: for any \( x \) and \( y \): \( x^{y \mod z} \mod n = x^{y \mod z} \mod n \)

 thus, \( c^d \mod n = (m^e \mod n)^d \mod n \)
  \[ = m^{ed} \mod n \]
  \[ = m^{ed} \mod n \]
  \[ = m^1 \mod n \]
  \[ = m \]

Why \( K_B(K_B^*(m)) = m = K_B^*(K_B(m)) \)?

follows directly from modular arithmetic:

\[ (m^e \mod n)^d \mod n = m^{ed} \mod n \]
\[ = m^{ed} \mod n \]
\[ = (m^d \mod n)^e \mod n \]
**Why is RSA secure?**
- Suppose you know Bob’s public key \((n,e)\). How hard is it to determine \(d\)?
- Essentially need to find factors of \(n\) without knowing the two factors \(p\) and \(q\).
  - Fact: factoring a big number is hard.

**RSA in practice: session keys**
- Exponentiation in RSA is computationally intensive.
- DES is at least 100 times faster than RSA.
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data.

**Protocol ap1.0:** Alice says “I am Alice”

**Authentication**
- **Goal:** Bob wants Alice to “prove” her identity to him.

**Protocol ap1.0:** Alice says “I am Alice”

**Protocol ap2.0:** Alice says “I am Alice” in an IP packet containing her source IP address.

**Authentication: another try**
- **Goal:** Bob wants Alice to “prove” her identity to him.

**Protocol ap1.0:** Alice says “I am Alice”

in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice.

**Authentication: another try**
- **Goal:** Bob wants Alice to “prove” her identity to him.

**Protocol ap1.0:** Alice says “I am Alice”

in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice.

**Authentication: another try**
- **Goal:** Bob wants Alice to “prove” her identity to him.

**Protocol ap1.0:** Alice says “I am Alice”
**Network Security**

**Authentication: another try**

**Protocol ap2.0**: Alice says “I am Alice” in an IP packet containing her source IP address.

Trudy can create a packet “spoofing” Alice’s address.

**Protocol ap3.0**: Alice says “I am Alice” and sends her secret password to “prove” it.

Alice’s IP addr | Alice’s password | I’m Alice
---|---|---
OK

Failure scenario??

**Authentication: yet another try**

**Protocol ap3.1**: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

Alice’s IP addr | Alice’s encrypted password | I’m Alice
---|---|---
OK

Playback attack: Trudy records Alice’s packet and later plays it back to Bob.

Failure scenario??

**Authentication: yet another try**

**Protocol ap3.1**: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

Alice’s IP addr | Alice’s encrypted password | I’m Alice
---|---|---
OK

Record and playback still works!

**Authentication: yet another try**

**Goal**: avoid playback attack

nonce: number (R) used only once-in-a-lifetime

**ap4.0**: to prove Alice “live”, Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key.

“I am Alice”

R

K_{a,b}(R)

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

**Failures, drawbacks?**
**Authentication: ap5.0**

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

**ap5.0:** use nonce, public key cryptography

```
“I am Alice”
```

```
Bob computes
K^B_A(K_A(R)) = R
```

and knows only Alice could have the private key, that encrypted R such that

```
K^B_A(K_A(R)) = R
```

**ap5.0: security hole**

*man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)

```
I am Alice
```

```
Bob computes
K^B_A(K_A(R)) = R
```

```
I am Alice
```

```
Trudy gets
m = K^A_T(K^B_A(m)) sends m to Alice encrypted with Alice’s public key
```

**Digital signatures**

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

**Digital signatures**

simple digital signature for message m:

- Bob signs m by encrypting with his private key K^B_A,

```
Bob’s message, m
```

creating “signed” message, K^B_A(m)

```
Bob signs m by encrypting with his private key K^B_A,
```

```
creating “signed” message, K^B_A(m)
```

- Alice verifies m signed by Bob by applying Bob’s public key

```
K^B_A to K_A(m) then checks K^B_A(K_A(m)) = m.
```

- If K^B_A(K_A(m)) = m, whoever signed m must have used Bob’s private key.

Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m’

**non-repudiation:**

- Alice can take m, and signature K^B_A(m) to court and prove that Bob signed m
Network Security

Message digests

- Computationally expensive to public-key-encrypt long messages

**Goal:**
- Fixed-length, easy-to-compute digital "fingerprint"
- Apply hash function $H$ to $m$, get fixed size message digest, $H(m)$.

**Hash function properties:**
- Many-to-1
- Produces fixed-size msg digest (fingerprint)
- Given message digest $x$, computationally infeasible to find $m$ such that $x = H(m)$

**Hash function:**

- MD5 hash function widely used (RFC 1321)
  - Computes 128-bit message digest in 4-step process.
  - Arbitrary 128-bit string $x$, appears difficult to construct msg $m$ whose MD5 hash is equal to $x$
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Internet checksum: poor crypto hash function

- Internet checksum has some properties of hash function:
  - Produces fixed length digest (16-bit sum) of message
  - Is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<table>
<thead>
<tr>
<th>Message</th>
<th>ASCII format</th>
<th>Message</th>
<th>ASCII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I O U</td>
<td>49 4F 55 31</td>
<td>I O U 9</td>
<td>49 4F 55 39</td>
</tr>
<tr>
<td>0 0 . 9</td>
<td>30 30 2E 39</td>
<td>0 0 . I</td>
<td>30 30 2E 31</td>
</tr>
<tr>
<td>9 B O B</td>
<td>39 42 D2 42</td>
<td>9 B O B</td>
<td>39 42 D2 42</td>
</tr>
</tbody>
</table>

B2 C1 D2 AC — different messages — B2 C1 D2 AC

but identical checksums!

Digital signature = signed message digest

**Bob sends digitally signed message:**

- Large message $m$
- $H(m)$
- Bob's private key $K_B$
- Digital signature = encrypted message digest $K_B(H(m))$

**Alice verifies signature, integrity of digitally signed message:**

- Bob's public key $K_B$
- Digital signature (decrypt) $H(m)$
- $H(m)$

Recall: ap5.0 security hole

**Man (or woman) in the middle attack:**
Trudy poses as Alice (to Bob) and as Bob (to Alice)

Public-key certification

**Motivation:**
- Trudy plays pizza prank on Bob
  - Trudy creates e-mail order: Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob.
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store

- Trudy sends to Pizza Store her public key, but says it's Bob's public key

- Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob

- Bob doesn't even like pepperoni
Certification authorities

- **certification authority (CA):** binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - Certificate containing E’s public key digitally signed by CA – CA says "this is E’s public key".

Bob’s public key

```
+-------------+              +-------------+              +-------------+
| Bob’s public key |               | CA public key |               | Bob’s public key |
| K_B            |               | K_CA         |               | K_B            |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
|                 |               |             |               |                 |
| digital signature (encrypt) | certificate for Bob’s public key, signed by CA | CA private key |
| CA              |               |             |               |                 |
```

when Alice wants Bob’s public key:
- gets Bob’s certificate (Bob or elsewhere).
- apply CA’s public key to Bob’s certificate, get Bob’s public key.