w7.1- Transport Layer

w7.1- Transport Layer.mp4 — Haruna Media Player

Review: The Magic of IP

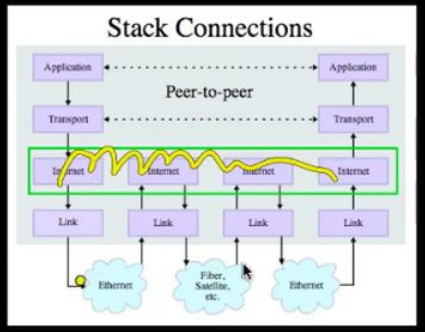
- What it does - Tries to get one packet across a 5-20 of hops from one network to another network
- Keeps track of the good and bad paths for traffic - tries to pick better paths when possible
- But no guarantee of delivery - if things go bad - the data vanishes
- This makes it fast and scalable - and ultimately "reliable" because it does not try to do too "everything"

packets to their destination network from one network to another network as well as

00:00:29 / 00:09:20

Internet Protocol

- So many links / hops
- So many routes
- Thinks can change dynamically and IP has to react (links up/ down)
- IP can drop packets



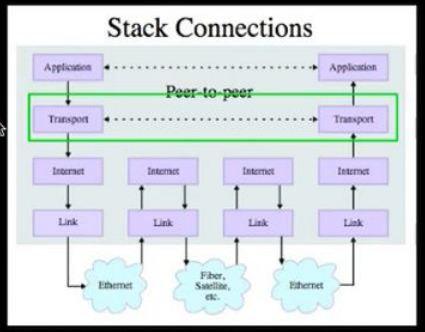
Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

paths, they can make use of all kinds of crazy links.

w7.1- Transport Layer.mp4 — Haruna Media Player

Transport Protocol (TCP)

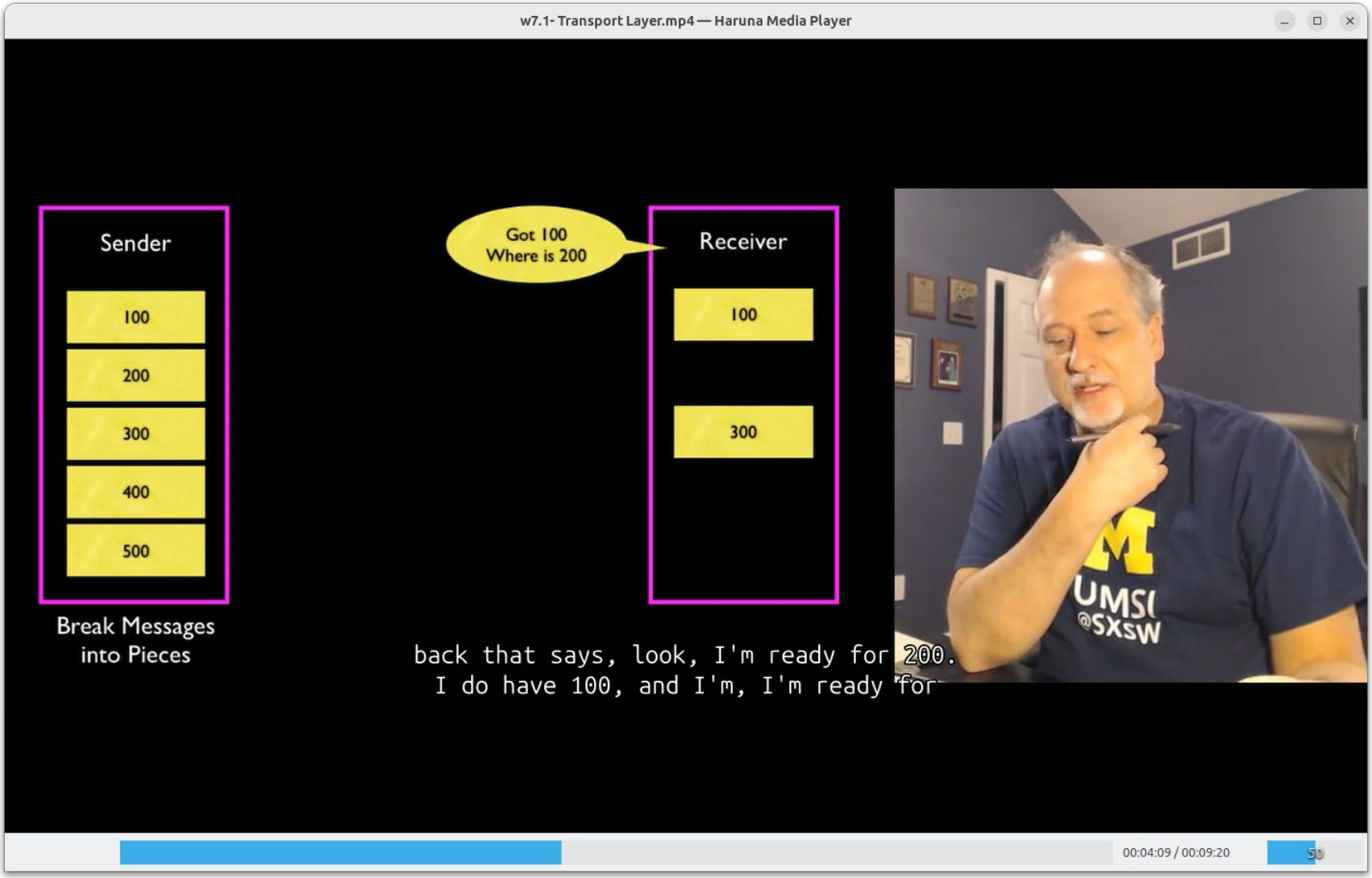
- Built on top of IP
- Assumes IP might lose some data
- In case data gets lost - we keep a copy of the data a we send until we get an acknowledgement
- If it takes "too long" - just send it again



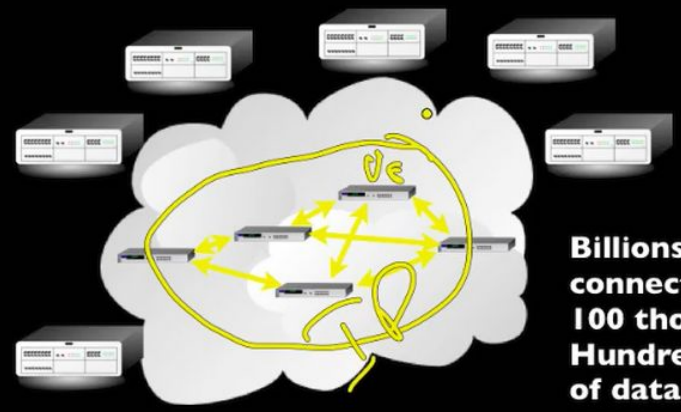
Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

complex.
The purpose of the TCP layer is to

00:01:33 / 00:09:20



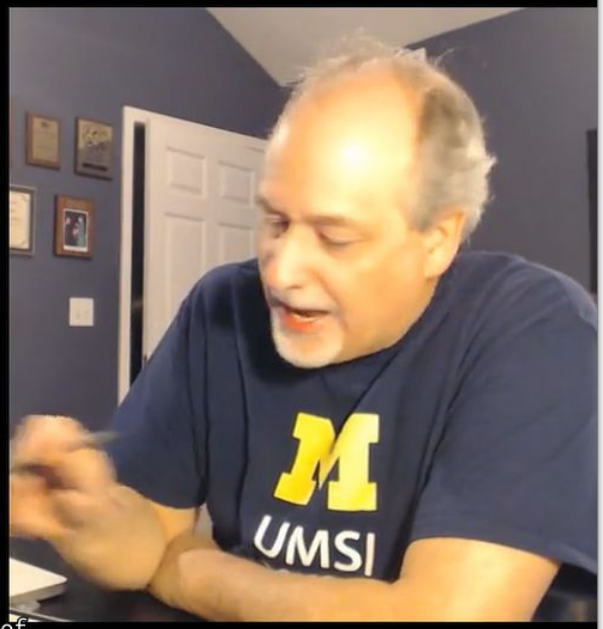
w7.1- Transport Layer.mp4 — Haruna Media Player



Clipart: http://www.clker.com/search/networksym/1

Billions of computers connected to the internet; 100 thousands of routers. Hundreds of billions bytes of data enroute at any moment.

Storage of enroute data done at the edges only!

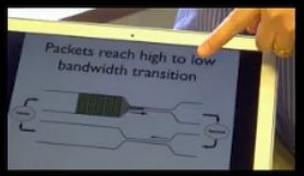


you know, all over the place.
Just, store up piles of packets, piles of

00:05:48 / 00:09:20

One (of many) Scary Problem(s)

- In 1987 as local campuses with 10 MBit networks were connected together using 56Kbit leased lines, things kind of fell apart
- At some point, when there was a little too much traffic, it all fell apart...



http://www.youtube.com/watch?v=1Vg1MeRYmWI
http://en.wikipedia.org/wiki/Van_Jacobson
http://en.wikipedia.org/wiki/TCP_congestion_avoidance_algorithm

And it turns out that there is still a lot of engineering to make that happen.

Summary

The provided transcript discusses the **Transport Layer** (specifically TCP) in the four-layer internet architecture and its role in making communication reliable.

The Role of the Transport Layer

The Transport Layer sits above the Link Layer (e.g., Ethernet) and the Internetwork Layer (IP). While IP is a "postcard" layer that moves packets quickly but without guarantees of perfection or order, TCP compensates for these potential errors.

Key Functions of TCP

- **Reliability through Acknowledgments:** Data is broken into packets and sent to the destination. The sender retains these packets until it receives an acknowledgment from the receiver; if a packet is lost, it is retransmitted until acknowledged.
- **Resource Management:** TCP determines if the underlying network is fast or slow to ensure efficient sharing of resources.
- **Speculative Sending:** To optimize network use, TCP "guesses" and sends a stream of packets rather than waiting for an acknowledgment after every single one.

The "Genius" of Internet Architecture

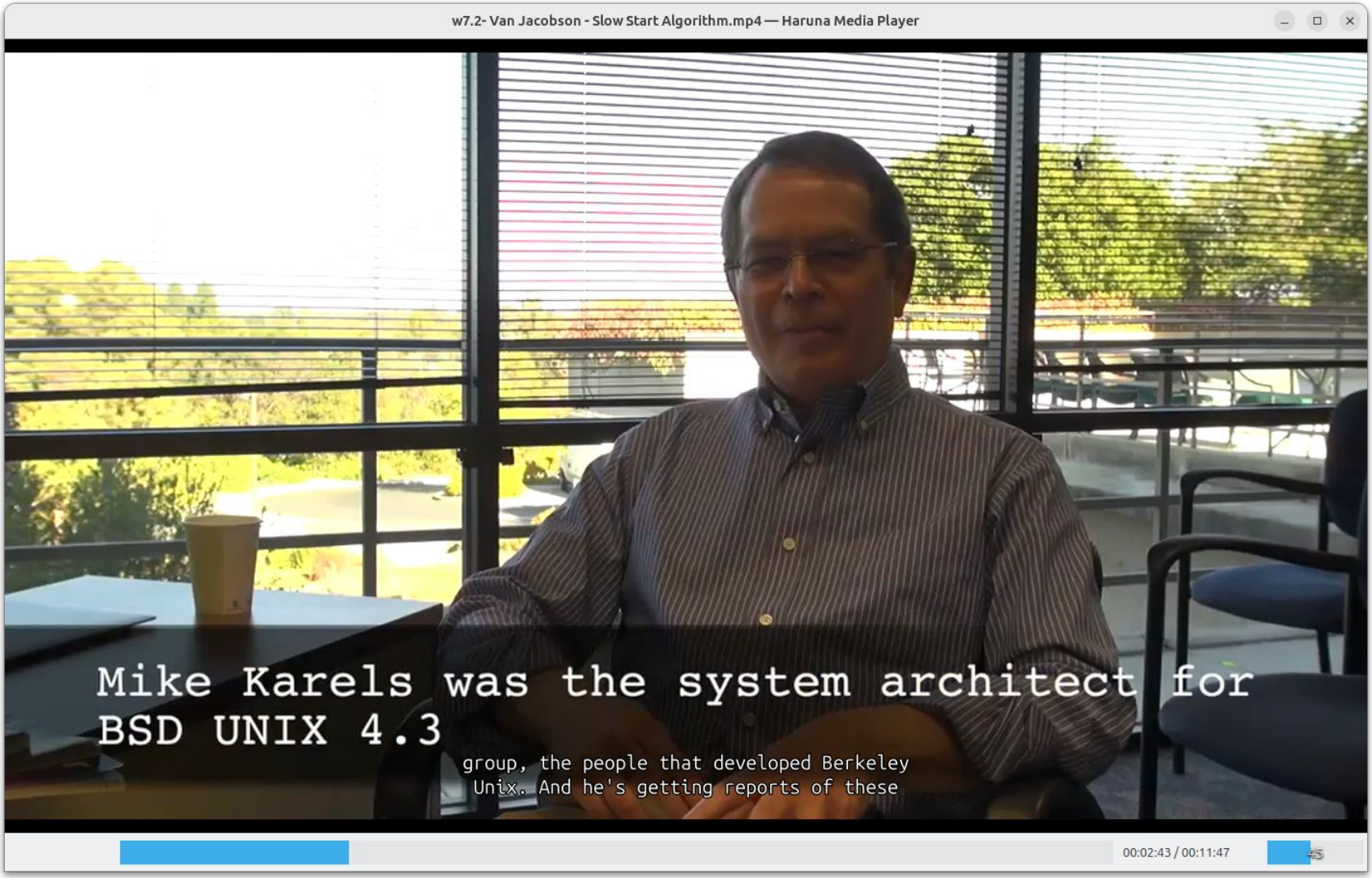
A fundamental design principle discussed is the placement of **long-term storage**.

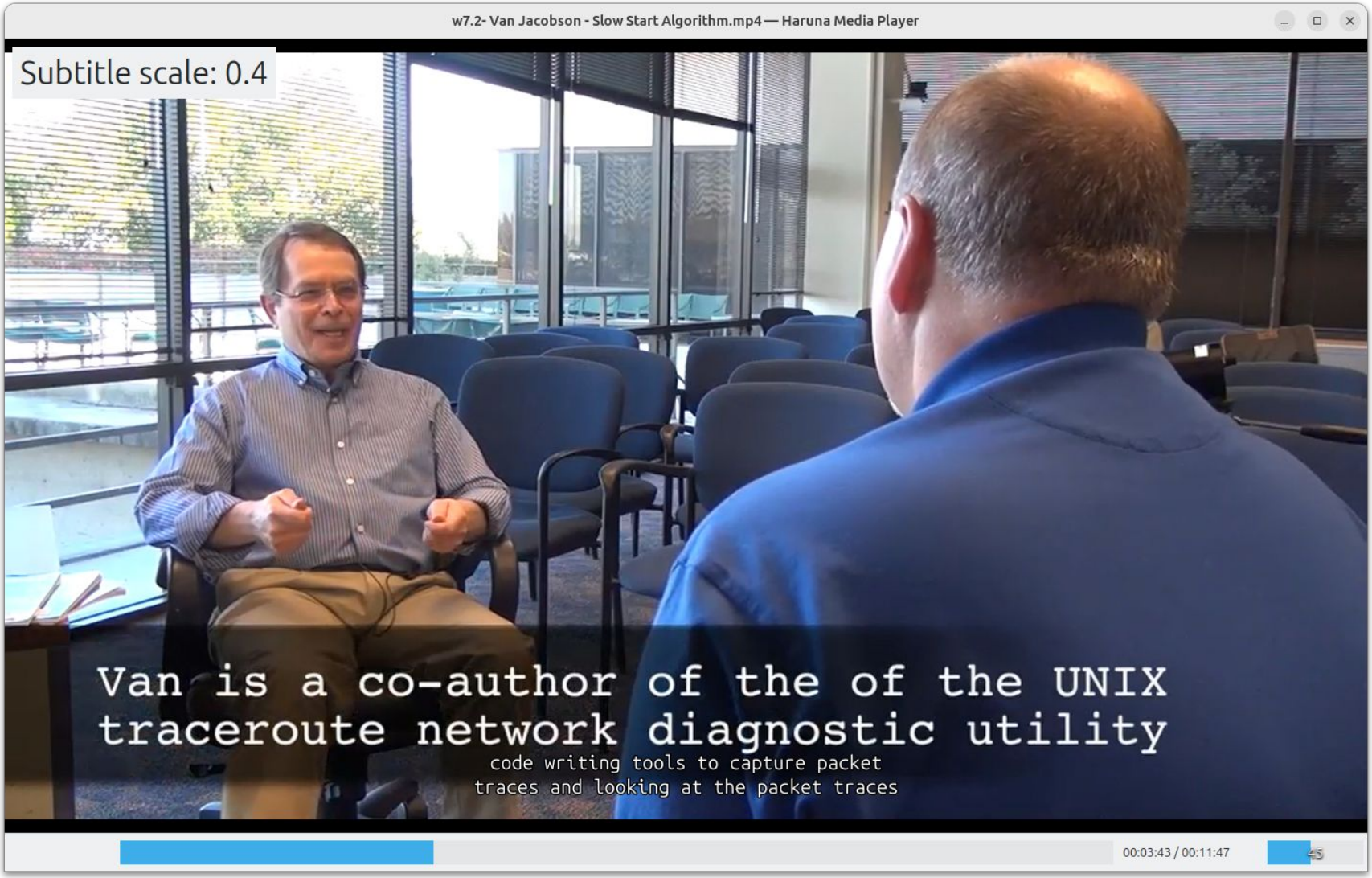
- **Agile Routers:** Routers in the middle of the network are designed to be fast and dynamic; they are not required to store packets long-term and are even encouraged to throw them away if the network is congested.
- **Edge Storage:** The responsibility for storing packets for retransmission lies with the billions of computers (laptops, phones) at the "outside" of the network.

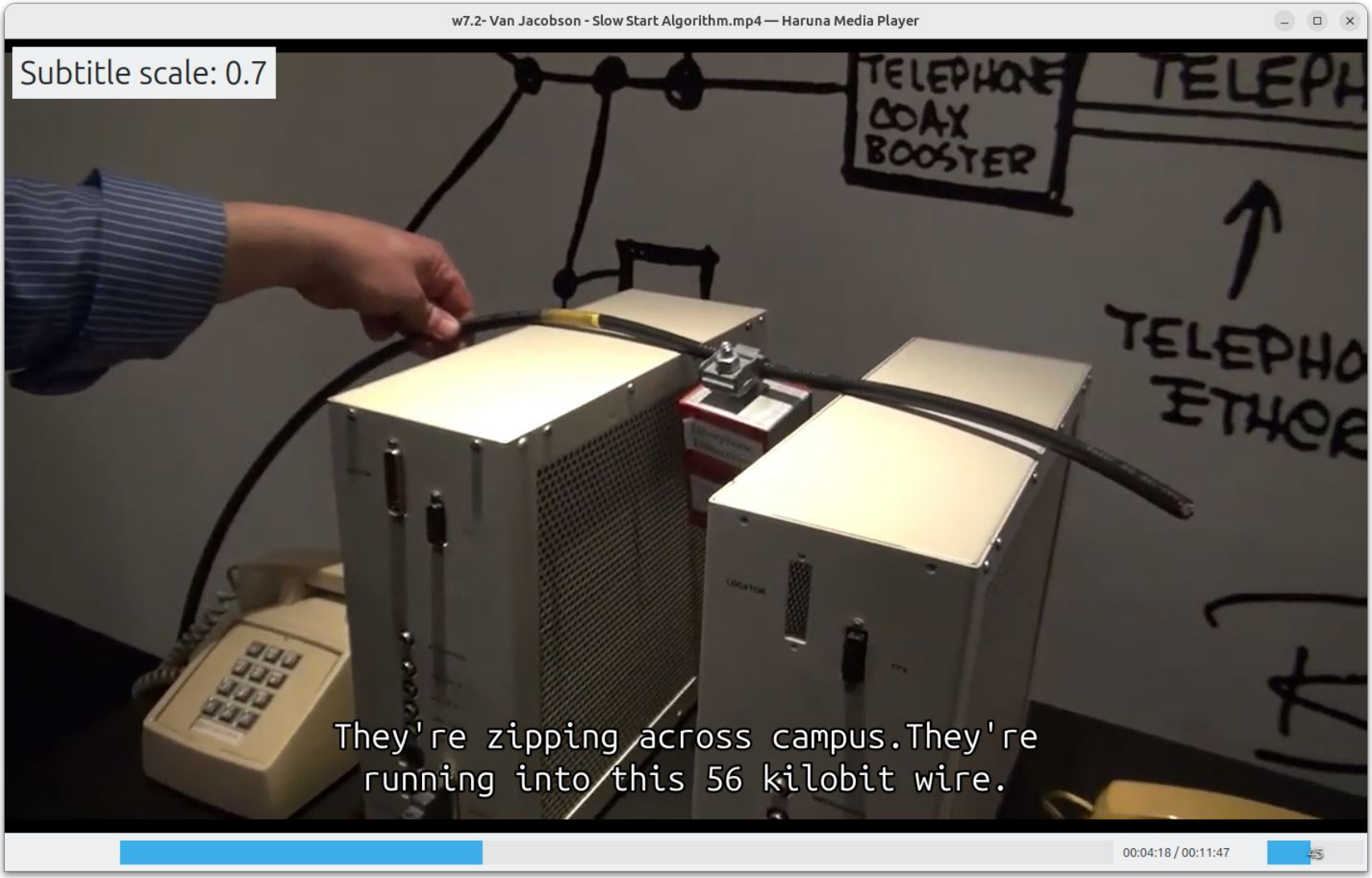
Historical Context: Van Jacobson

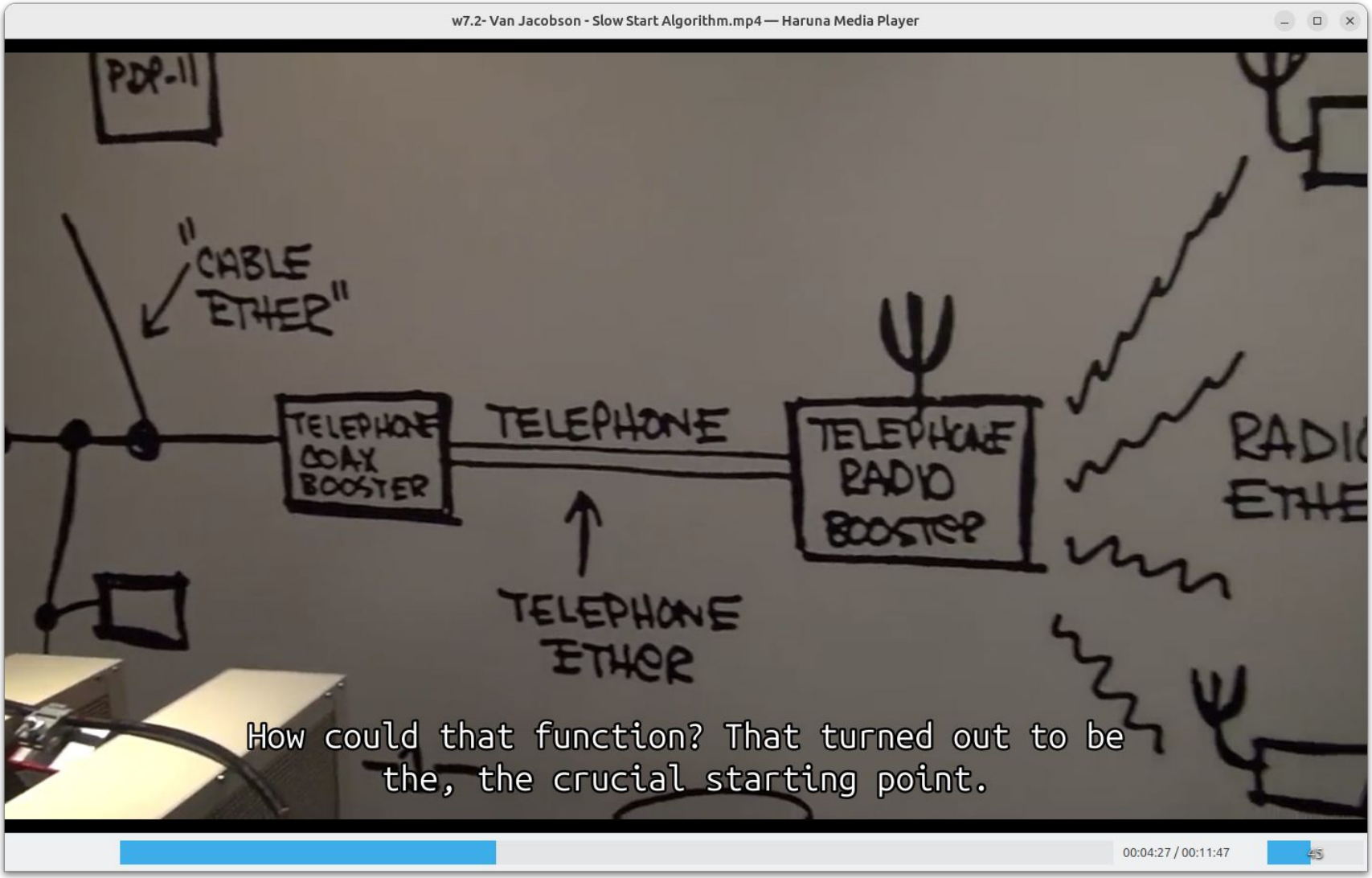
The transcript credits **Van Jacobson** with saving the internet in the late 1980s when the network backbone was failing under increased load. He developed the **slow start algorithm**, a congestion control method now found in every TCP implementation to prevent network crashes.

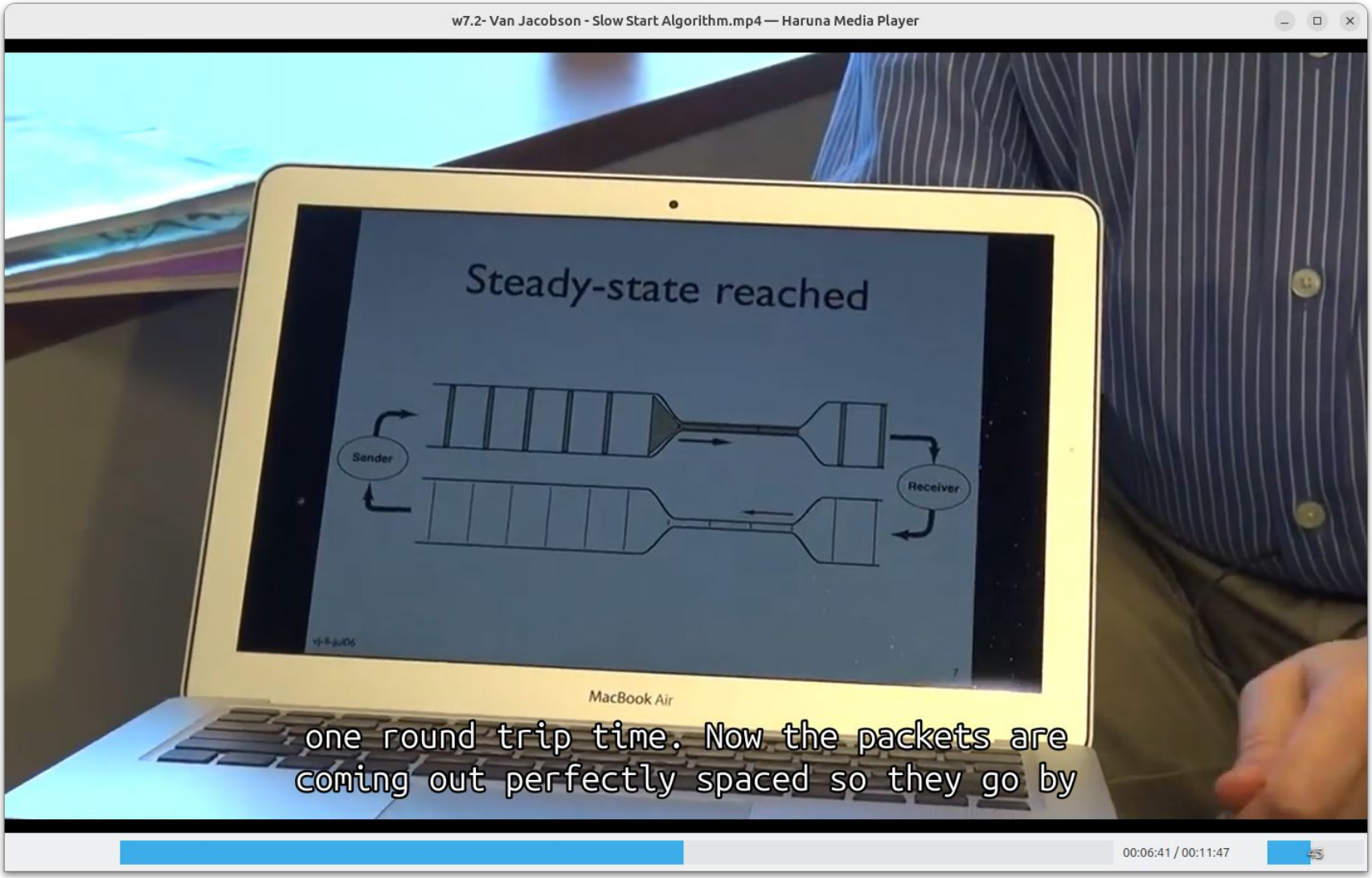
w7.2- Van Jacobson - Slow Start Algorithm



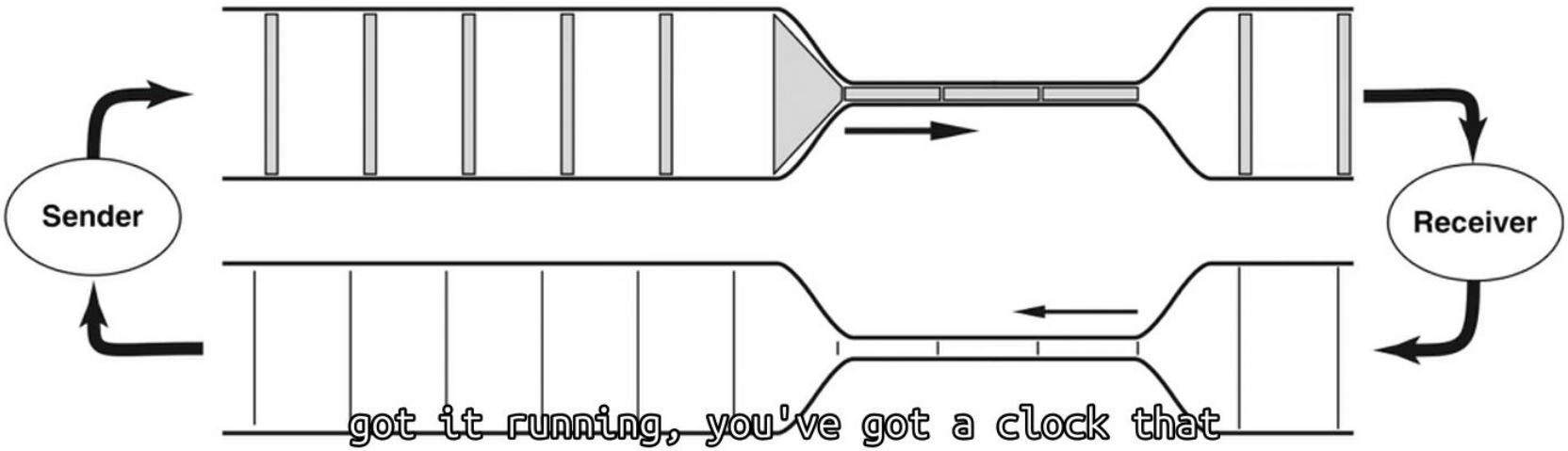




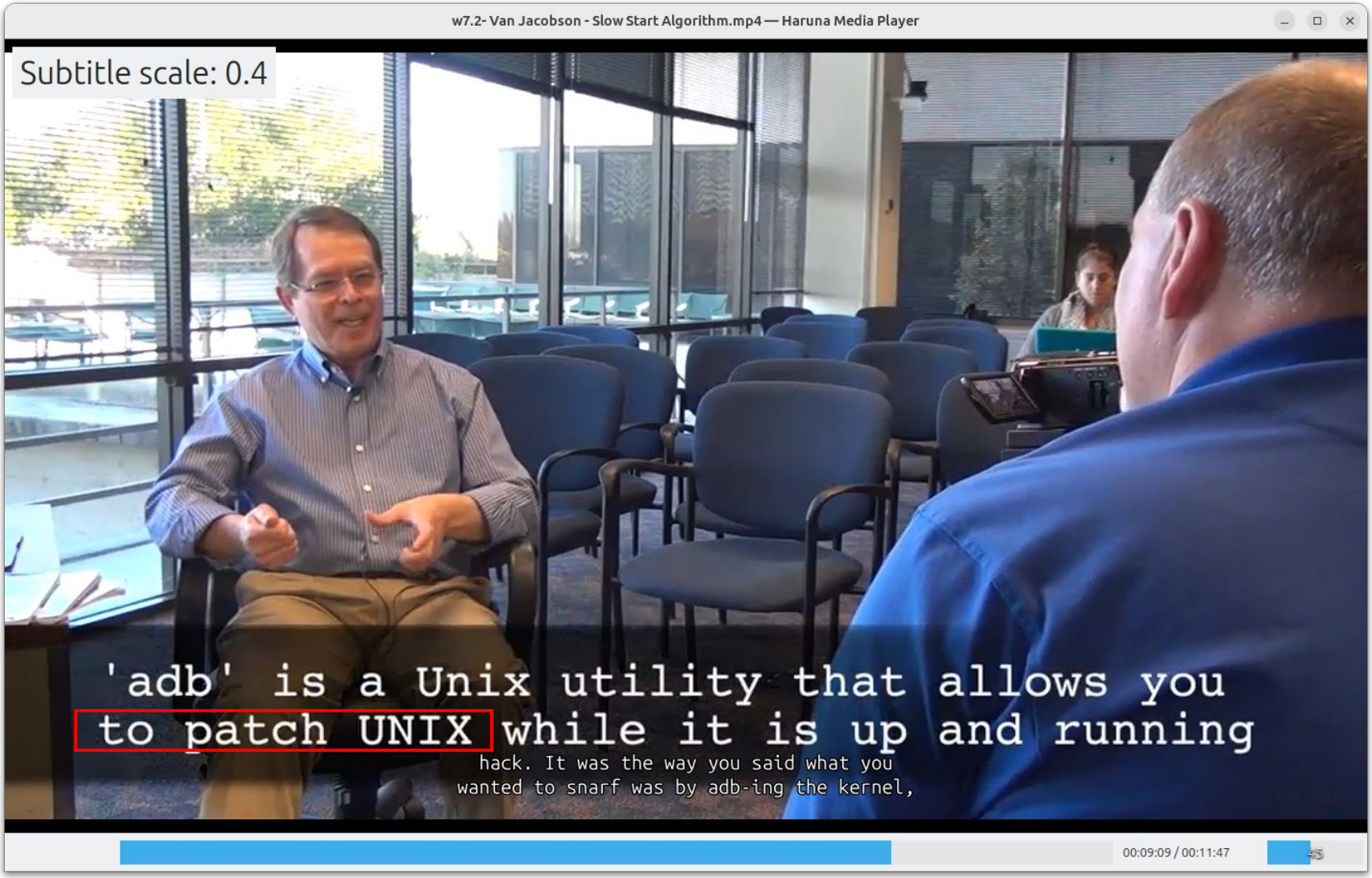


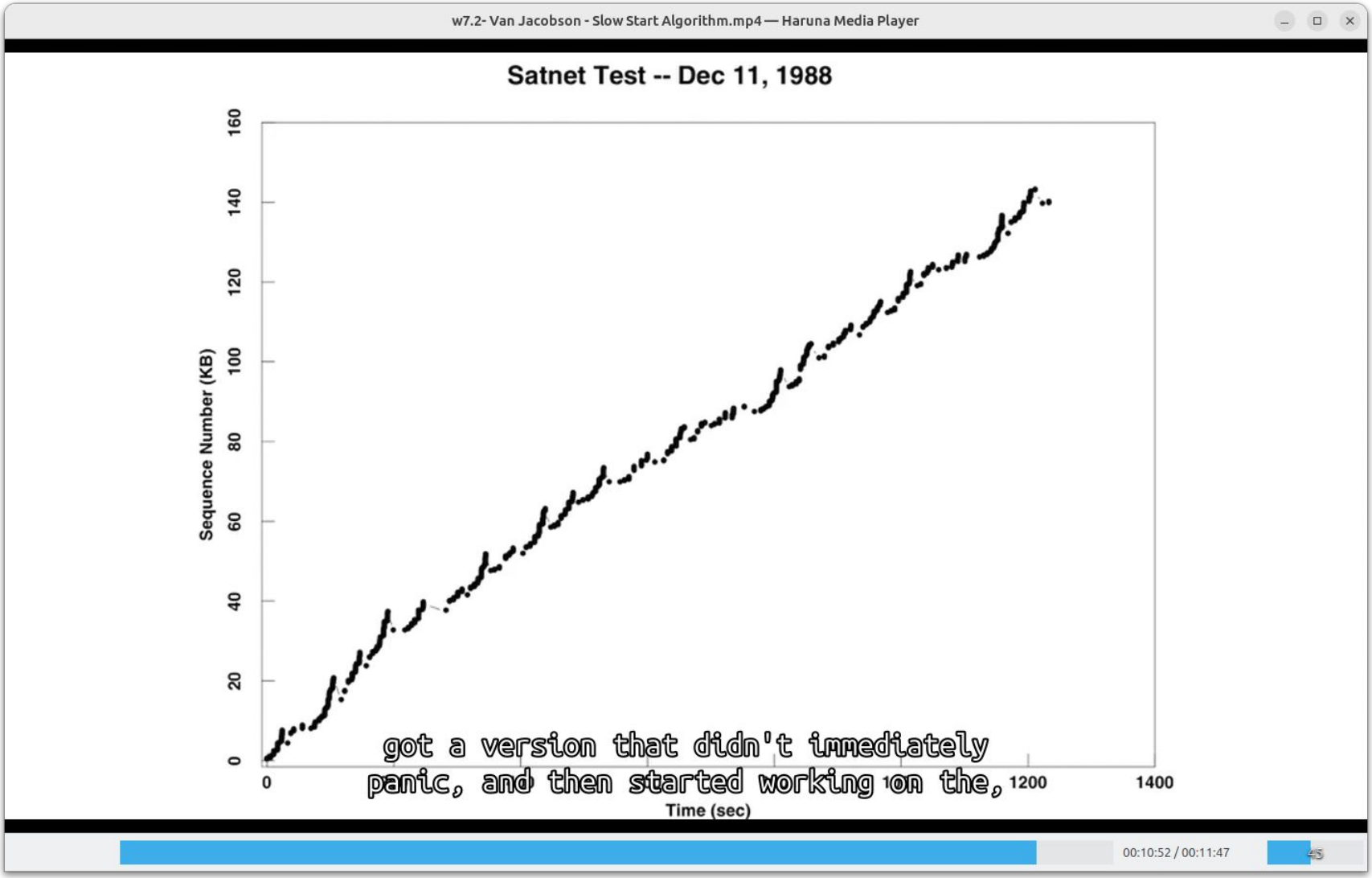


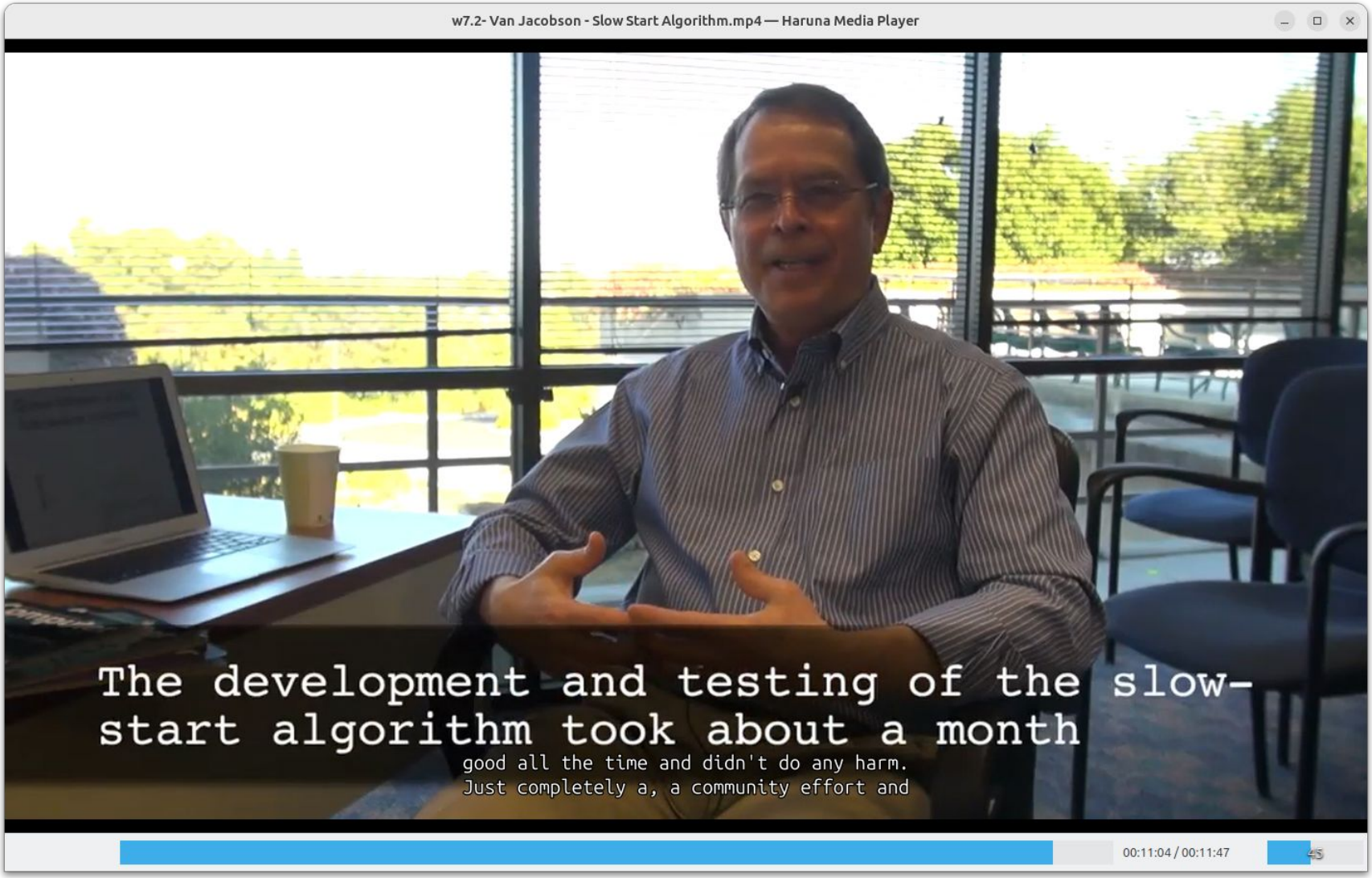
Steady-state reached

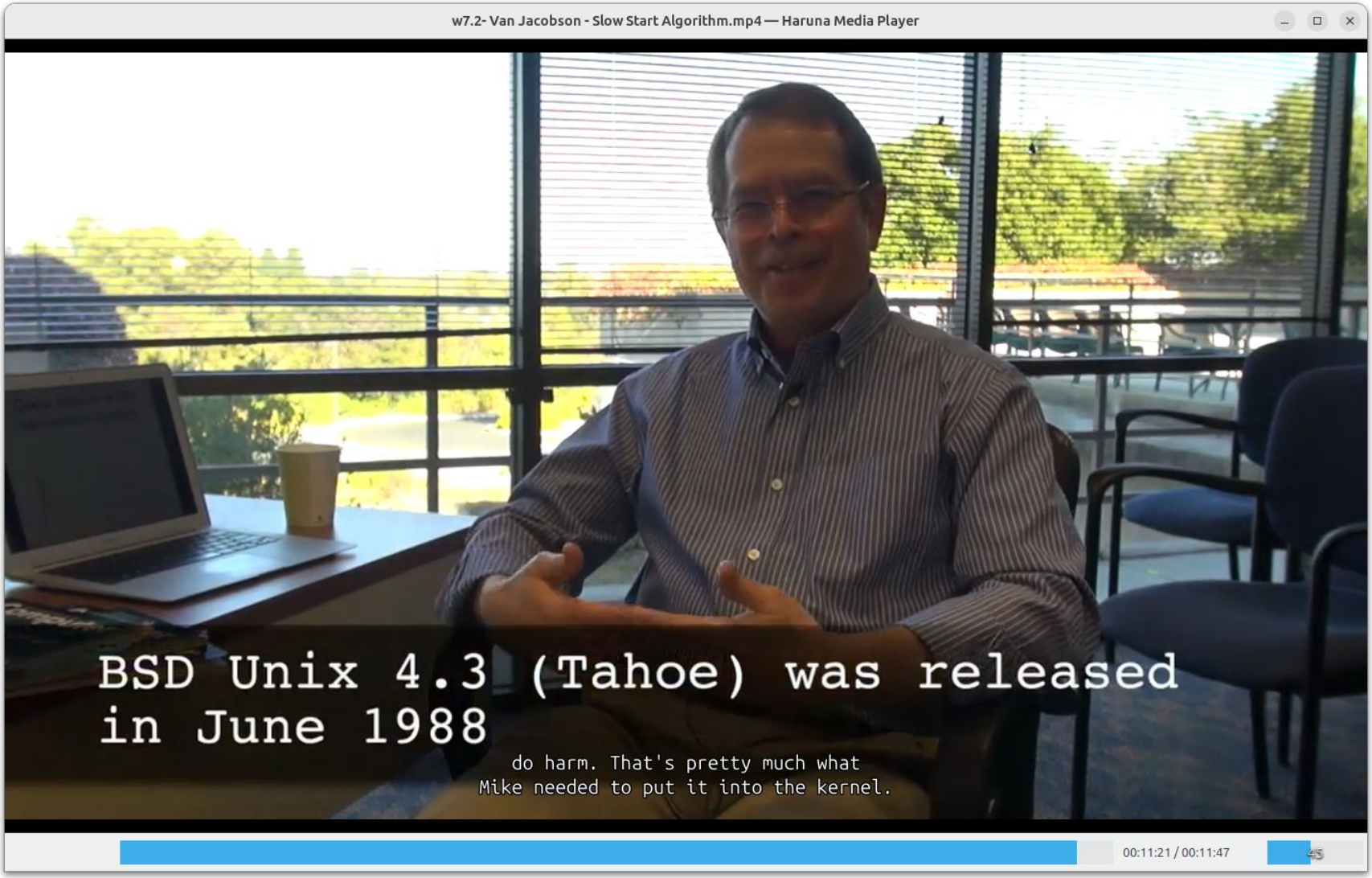


got it running, you've got a clock that
tells you exactly what to do. If you turn









The **Slow Start Algorithm** (慢启动算法) is a fundamental part of the TCP (Transport Control Protocol) congestion control mechanism. Its primary goal is to prevent a network from failing by ensuring a sender doesn't overwhelm the available bandwidth before it understands the network's capacity.

The Core Problem: Congestion Collapse

In the late 1980s, the internet faced "congestion collapse" because high-speed local networks were dumping massive amounts of data into much slower long-distance lines.

- When a connection starts, there is no "clock" (feedback) to tell the sender how fast the network can go.
- If a sender immediately sends a large burst of data, it saturates the buffers in the routers (gateways), leading to dropped packets and repeated failures.

How Slow Start Works

Invented by **Van Jacobson**, this algorithm allows the sender to grow the volume of data gradually.

- **Initialization:** The sender starts by sending a very small number of packets (originally just one).
- **The Feedback Loop:** The sender waits for an **Acknowledgment (ACK)** from the receiver.
- **Exponential Growth:** For every ACK received, the sender increases the number of packets it can have "in-flight". This causes the transmission rate to double with every round-trip time (RTT).
- **Reaching Steady State:** This continues until the sender reaches a specific threshold or a packet is lost. At that point, it transitions into a more cautious "Congestion Avoidance" phase.

Summary

This transcript features **Van Jacobson**, a key figure in the history of the internet, describing the "Congestion Collapse" of 1986 and how he and his colleagues engineered a solution that still powers our networks today.

Here is a summary of the events and technical breakthroughs described:

1. The Crisis: Congestion Collapse

In the mid-1980s, university campuses were connecting their high-speed (10 Mbps) local Ethernets to each other using the NSFNET's significantly slower 56 Kbps lines.

- **The Problem:** The massive mismatch in speed caused packets to pile up and drop at the gateways.
- **The Result:** Throughput dropped to near zero—sometimes only one packet every ten minutes—threatening the viability of the entire experimental internet.

2. The "Aha!" Moment: Self-Clocking

After months of analyzing packet traces, Jacobson realized that TCP could act as its own "clock".

- **The Bottleneck:** When packets from a fast network hit a slow wire, they naturally "spread out" in time.
- **The ACKs:** Once those packets reach the destination, the receiver sends back Acknowledgments (ACKs). These ACKs return to the sender with the exact spacing of the slowest point in the network.
- **The Result:** By waiting for an ACK before sending a new packet, the sender naturally matches the speed of the bottleneck, creating a "per-packet clock" that prevents overloading the network.

3. The Solution: Slow Start

The main difficulty was starting the connection when no "clock" (ACKs) yet existed.

- **Old Way:** Senders would "dump" a full window of packets immediately, saturating buffers and causing immediate failure.
- **Jacobson's Fix:** He introduced **Slow Start**, where a sender begins with a very small number of packets and grows the volume gradually as ACKs return. This allows the "clock" to synchronize before the network is overwhelmed.

4. Community Deployment

Since the internet was an experimental community effort at the time, the fix was deployed rapidly through a collaborative "cycle":

- Jacobson released patches via the TCP/IP mailing list.
- Users worldwide tested the code, sent back "kernel core dumps" when it crashed, and Jacobson fixed the bugs in real-time.
- Once the community agreed the code "mostly does good and never seems to do harm," it was rolled into the official **Berkeley Unix (BSD)** release, becoming the global standard.

w7.3- The Domain Name System

w7.3- The Domain Name System.mp4 — Haruna Media Player

Volume: 50

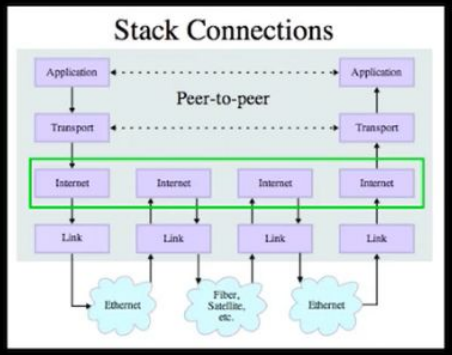
Domain Name System

The Domain Name System convert user-friendly names, like

www.umich.edu

to network-friendly IP addresses, like

141.211.32.166



Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

friendly names like www.umich.edu to, to
like a network friendly address.

00:00:12 / 00:07:37

Domain Name System

- Numeric addresses like 141.211.63.45 are great for Internet routers but lousy for people
- Each campus ends up with a lot of networks (141.211.*.*, 65.43.21.*)
- Sometimes (rarely) the IP address numbers get reorganized
- When servers physically move they need new IP addresses

it works, because domain names are what
we use all the time and IP addresses are

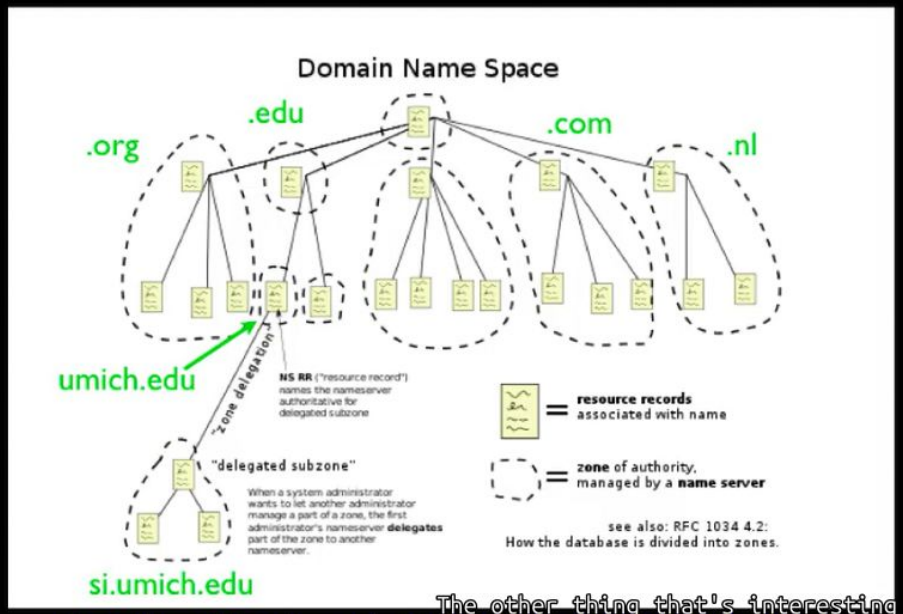
Subtitle scale: 0.4

DNS: Internet Address Book

- The Domain Name System is a big fast distributed database of Internet names to Internet "phone numbers"
- IP Addresses reflect technical "geography"
  - 141.211.63.44 - read left to right like a phone number
- Domain names reflect organizational structure
  - www.si.umich.edu - read right to left like postal address
  - 2455 North Quad, Ann Arbor, MI 48109, USA, Earth

uses caching so that it's locally fast
even if the network is partially down.

w7.3- The Domain Name System.mp4 — Haruna Media Player



http://en.wikipedia.org/wiki/File:Domain-name-space.svg

The other thing that's interesting about domain names is that they're owned.

00:03:58 / 00:07:37

Summary

The **Domain Name System (DNS)** acts as the internet's "address book," translating human-friendly names into the numeric **IP addresses** used by computers and routers. While it doesn't fit perfectly into a single layer of the four-layer architecture, it functions as a critical add-on that enables transparent mapping.

How DNS Works

- **Translation:** DNS converts names like `www.umich.edu` into network-friendly IP addresses.
- **Distributed Database:** It is a massive, fast, and distributed database that uses **caching** to remain functional even if parts of the network are down.
- **Independence:** Routers do not recognize domain names; they only move data based on IP addresses, which encode the technical geography of the internet.

Addressing Structures

The hierarchy of these addresses reads in opposite directions:

System	Direction	Hierarchy Example
IP Address	Left to Right	Network number (Physical attachment point) → Internal attachment point
Domain Name	Right to Left	`.edu` (Educational) → `umich.edu` (University) → `si.umich.edu` (School)

Ownership and Hierarchy

Domain names are owned and managed from **right to left** through a tiered authority system.

- **Top-Level Domains (TLD):** Organizations like **Educause** own the `.edu` domain in the public trust and award subdomains to accredited institutions.
- **Commercial Domains:** Domains like `.com` or `.org` are generally first-come, first-served, though trademark laws can influence ownership in cases of "legitimate purpose".
- **Subdomain Management:** Once an organization owns a domain (e.g., `umich.edu`), it can create its own committees to award sub-subdomains (e.g., `si.umich.edu`) to internal groups.

w7.4- TCP Wrap Up

w7.4- TCP Wrap Up.mp4 — Haruna Media Player

Subtitle scale: 0.6

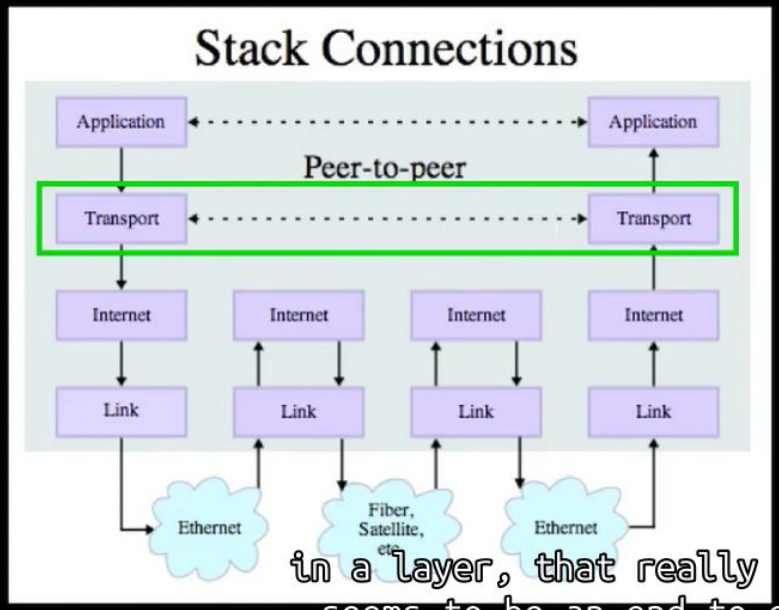
Transport Protocol (TCP)

- The responsibility of the transport layer is to present a reliable end-to-end pipe to the application
- Data either arrives in the proper order or the connection is closed
- TCP keeps buffers in the sending and destination system to keep data which has arrived out of order or to retransmit if necessary
- TCP provides individual connections between applications

So basically the transport control protocol has a responsibility of

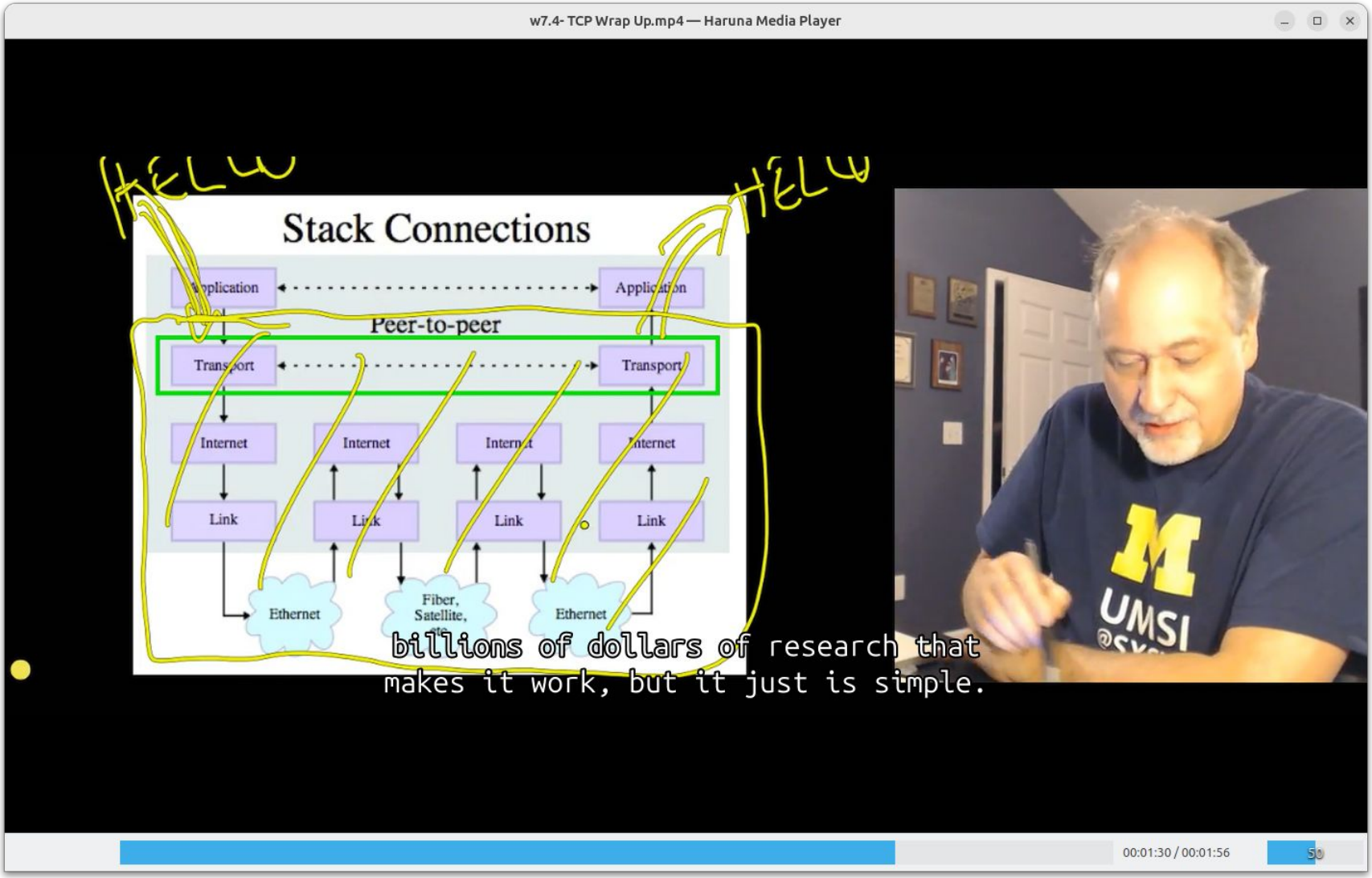
00:00:04 / 00:01:56

w7.4- TCP Wrap Up.mp4 — Haruna Media Player



in a layer, that really provides us what seems to be an end to end connection.

00:00:32 / 00:01:56



Summary

The provided transcript summarizes the primary function of the **Transport Control Protocol (TCP)** within the layered internet architecture and how it creates a reliable environment for applications.

The Responsibility of TCP

TCP is designed to handle the "imperfections" of the underlying IP layer. Its core duties include:

- **Compensating for Errors**: TCP manages data that may arrive out of order or not arrive at all.
- **Data Marking and Storage**: The protocol marks data and stores it on the source computers until the destination computer sends an acknowledgment.
- **Edge Buffering**: By keeping data buffers on the "edge" (the source and destination computers) rather than inside the network, the internet is able to grow more effectively.

The "End-to-End" Illusion

The result of this complex bookkeeping is a layer that provides what appears to be a simple, seamless connection.

- **Data Streams**: An application can "roll in" a stream of data at one end and expect a reliable stream to come out the other side.
- **Reliability and Order**: Data is guaranteed to arrive reliably and in the same order it was sent.
- **Abstraction of Complexity**: The goal of this layered architecture is to allow applications to "pretend" the complexity of the internet is simple.