Resilient Connections for SSH and TLS

Teemu Koponen and Pasi Eronen, HIIT
Mikko Särelä, Nokia Research Center
Presented by Chakchai So-In (cs5@cse.wustl.edu)

Outline

- **Introduction**
- Related work and Goals
- Design principles
- Protocol extensions
- Implementation considerations
- Evaluation
- Conclusions

Motivation

- Applications have longer uptimes and experience more disruptions.
- Laptop Suspension/Hibernation (long disconnection)
- Switching multiple network interface
- User changes location (change of IP)

Cause the termination of the connection/ application!

Resilient connection

- Resilient connection → Session continuity
  - Continue the same session caused by Disconnection
- Implementing at higher layers desirable:
  - Long disconnection periods
  - No network infrastructure required and modified
  - Application get upgraded

Session layer is the lowest layer to implement resilient connection

What is SSH?

- SSH (Secure Shell defined in RFC 4346):
  - Session protocol for secure remote login and other secure network services
  - Provides user/ server authentication and encryption and data integrity protection

What is TLS?

- TLS (Transport Layer Security defined in RFC 2246):
  - Provides communication session privacy (encryption, authentication, and data integrity) over the Internet (WWW)
  - TLS and SSL are most recognized as the protocols that provide secure HTTP (HTTPS) for Internet transactions.
  - SSL (Secure Socket Layer) was developed by Netscape to secure transactions over WWW.
  - Then IETF developed a standard protocol that provided the same functionality. They used SSL 3.0 as the basis for that work, which became the TLS protocol.

Outline

- Introduction
- **Related work and Goals**
- Design principles
- SSH/TLS extension procedure
- Implementation considerations
- Evaluation
- Conclusions

Related work

- Several session continuity on “socket API libraries”
  - E.g. Persistent connections, Mobile TCP socket, Mobile-Socket, Reliable sockets, and Migrate
  - Library placed below socket API: virtual single unbroken connection to real TCP connections
  - Key difference: Deployability
    - Use of out-of-band signaling (separate TCP or UDP session) may require changes in network
    - Separate key exchange to protect the signaling messages -> additional overhead

Goals

- Develop resiliency extensions for the common TLS and SSH protocols
  - Real implementation are in OpenSSH and PureTLS.
  - Emphasize on deployability
- Analyze implementation issues faced when adding the extensions to existing software packages

Outline

- Introduction
- Related work and Goals
- **Design principles**
- SSH/TLS extension procedure
- Implementation considerations
- Evaluation
- Conclusions

Design Principals

- No network changes required
- Incremental deployment possible
- Limited end-point changes (only SSH/TLS)
- Long disconnections supported (>10sec)
  - No handover optimization
- In overall, emphasize on **deployability**

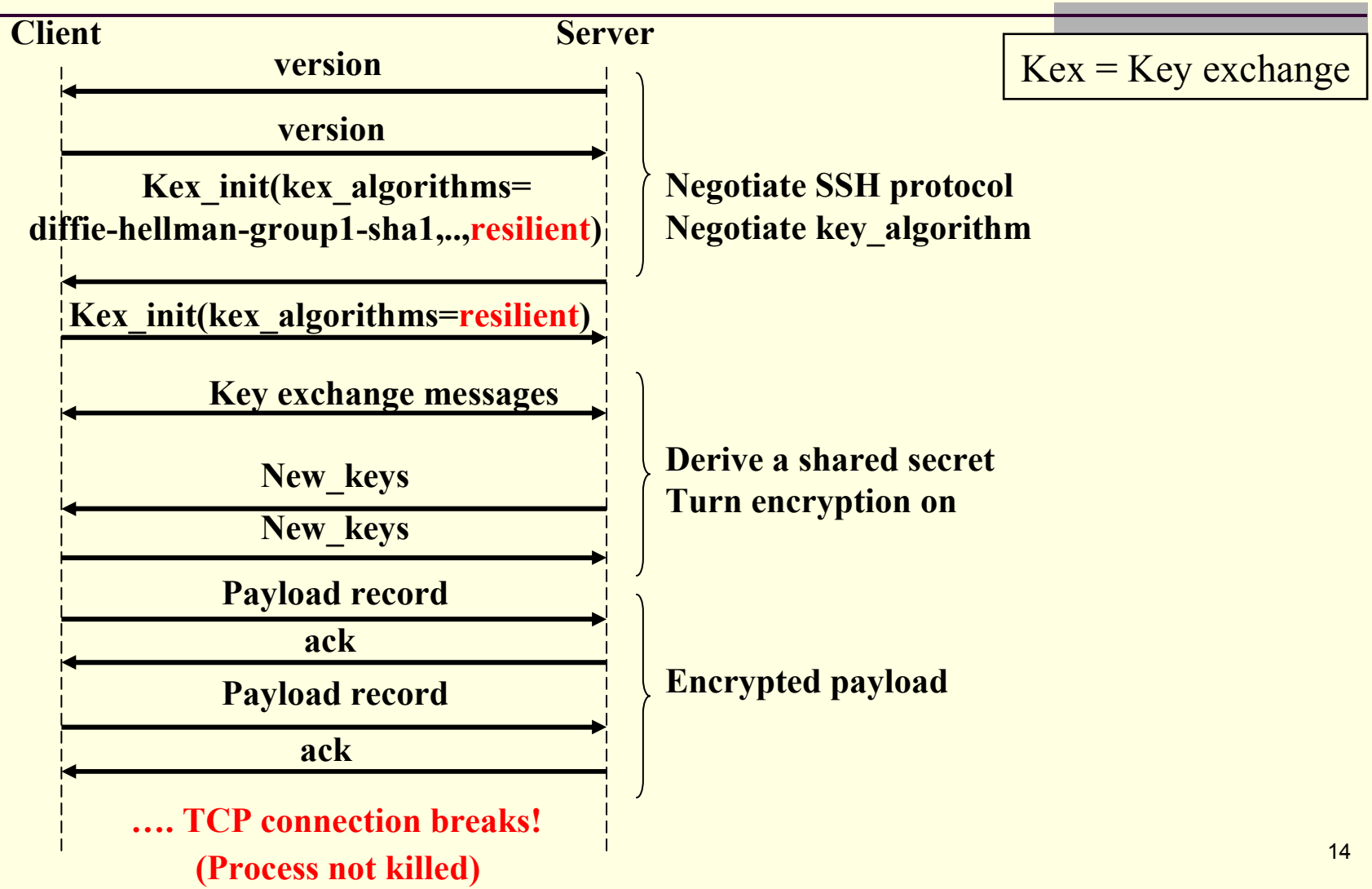
Supporting Deployability

Key Feature	***Implication***
In-band signaling	Works without changes in current middleboxes
Extension negotiation	Incremental deployment
Only application changed	No infrastructure required
Buffering and explicit ACKs	OS independency
Closing message	Determine if disconnection/ connection termination

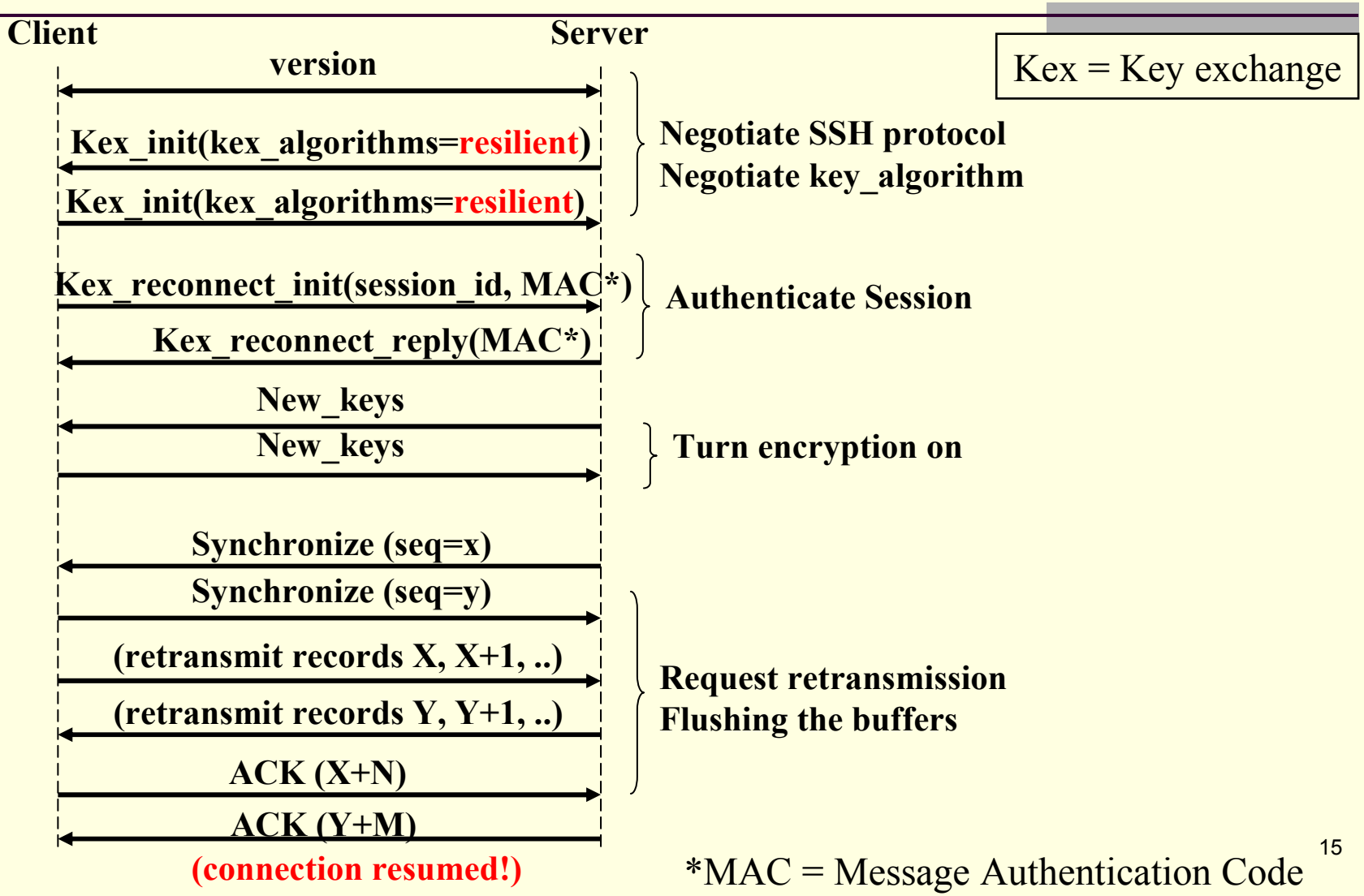
Outline

- Introduction
- Related work and Goals
- Design principles
- **SSH/TLS extension procedure**
- Implementation considerations
- Evaluation
- Conclusions

SSH procedure (initial)



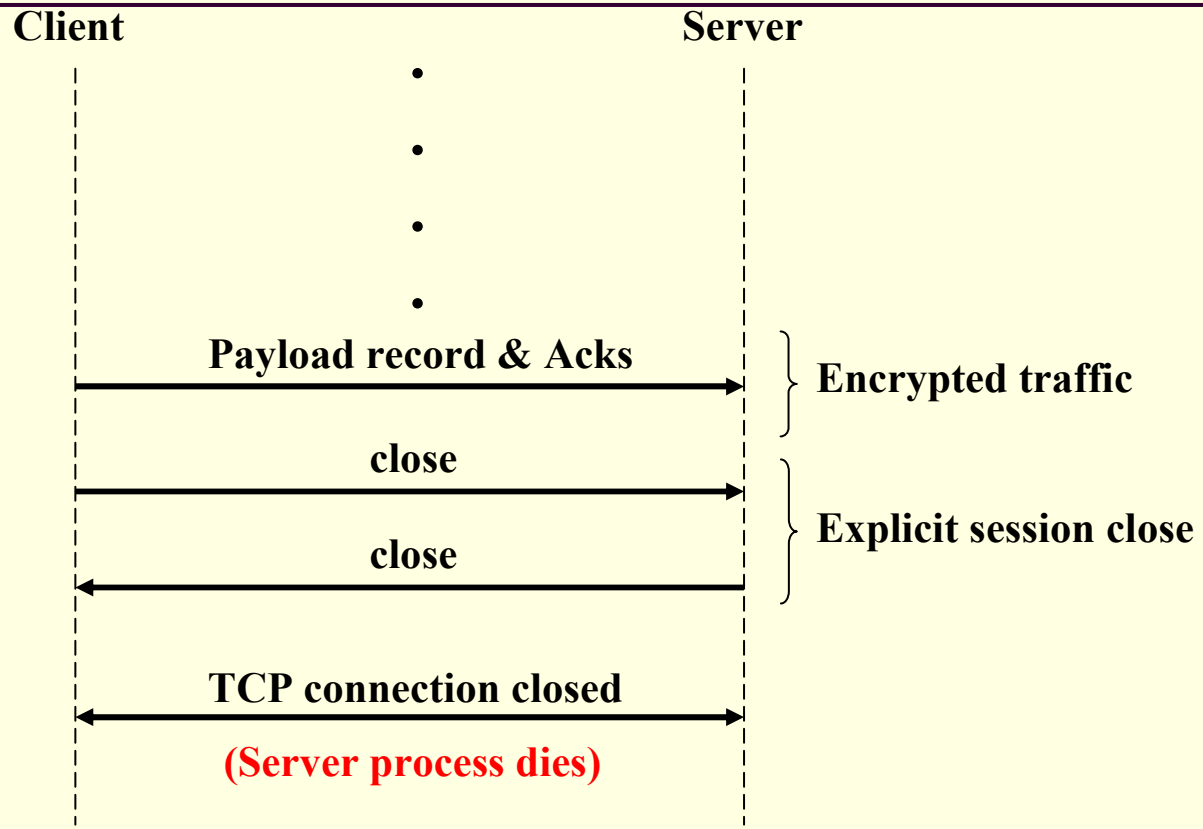
SSH procedure (reconnection)



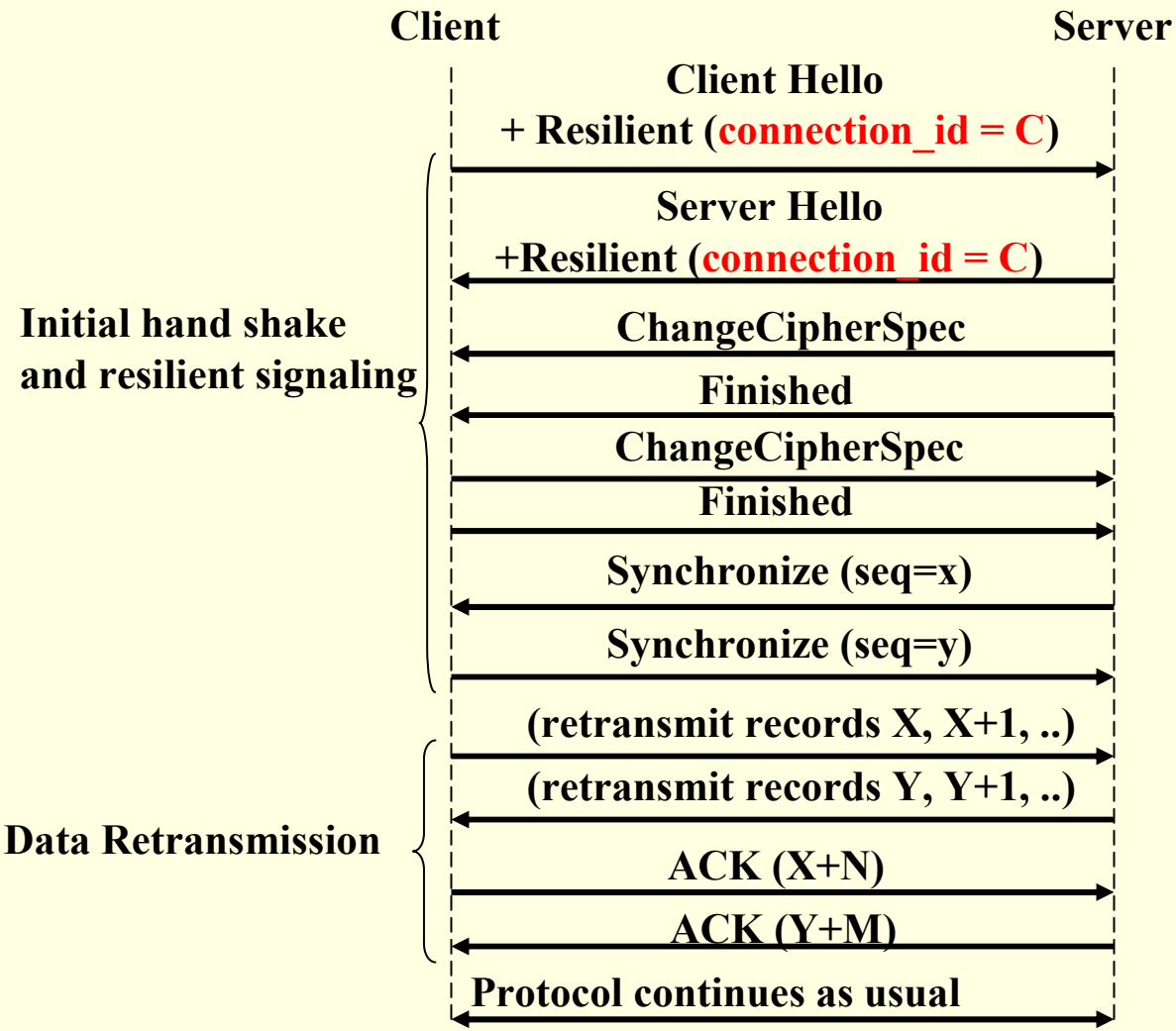
Kex = Key exchange

*MAC = Message Authentication Code

SSH procedure (closing)



TLS procedure (reconnection)



Security analysis

- Claim: no loss of security
- Shared keys created during the initial SSH or TLS protocol exchange
  - An attacker cannot spoof or modify the reconnect messages.
- Replay attacks are not possible
  - SSH and TLS key exchange messages include fresh ***nonces*** that are covered by a MAC later during the handshake.

Outline

- Introduction
- Related work and Goals
- Design principles
- SSH/TLS extension procedure
- **Implementation considerations**
- Evaluation
- Conclusions

Client considerations

- When to reconnect/ which interface to use?
  - A manual request, automatically, or preferred network interface becomes available
  - Rely on the operating system's source address selection
    - OpenSSH: A routing socket to monitor routing table changes
    - PureTLS: Polling the OS in regular intervals for the preferred interface

Server considerations

- When is a session discarded?
  - Configurable server-wide timeout (based on local policy)
- What is the server process strategy?
  - Create a new process for each new client connection
  - Maintain table mapping sessions for inter-process communication
  - New process passes state information to the corresponding old process
    - New process has less state to pass.
    - In practice, the new process information is easier to transfer.

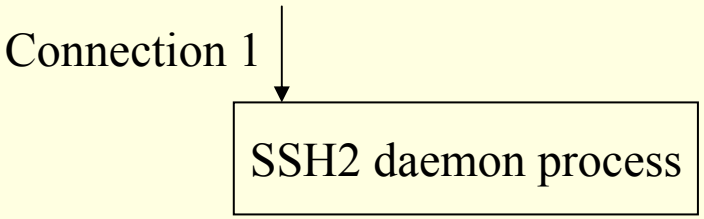
Atomic reconnections

- The protocol state machine of an old connection must not become corrupted if an attempt fails.
- New process transfers the state (a file descriptor / sequence numbers) to the old process only after a reconnection request is valid (by using keys in the old process.
  - Server: if a reconnection request is invalid, the old process sees nothing.
  - Client: either reconnection attempt succeeds or no state is affected.

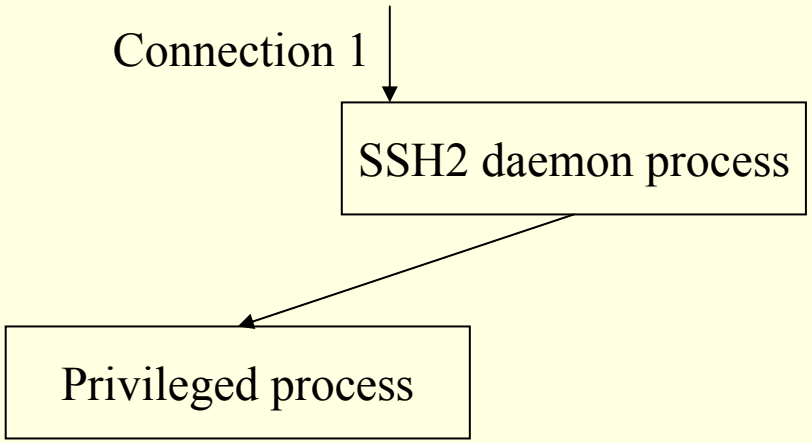
OpenSSH/PureTLS considerations

- OpenSSH
  - Require **state serialization and passing** across process boundaries
- PureTLS
  - The complexity are from **the requirement to keep the objects visible to the application** unchanged over reconnections e.g. additional layer of indirection for Socket.

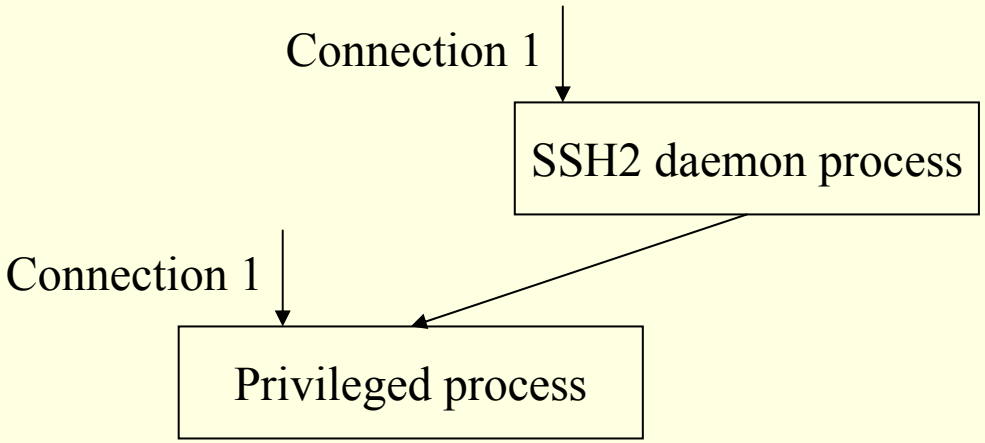
SSH process



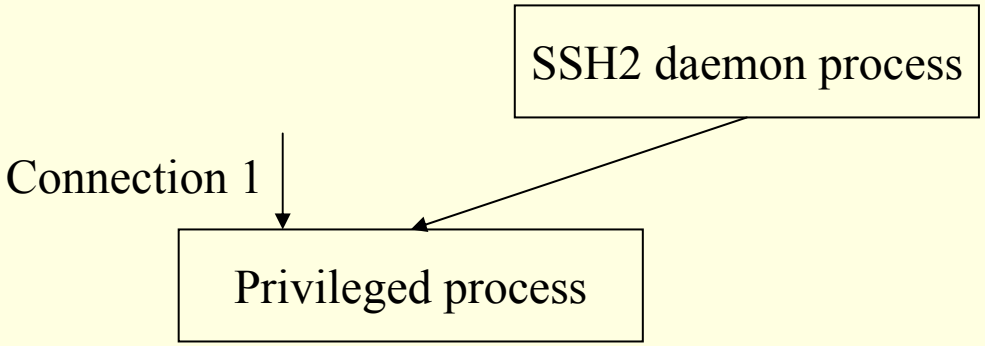
SSH process



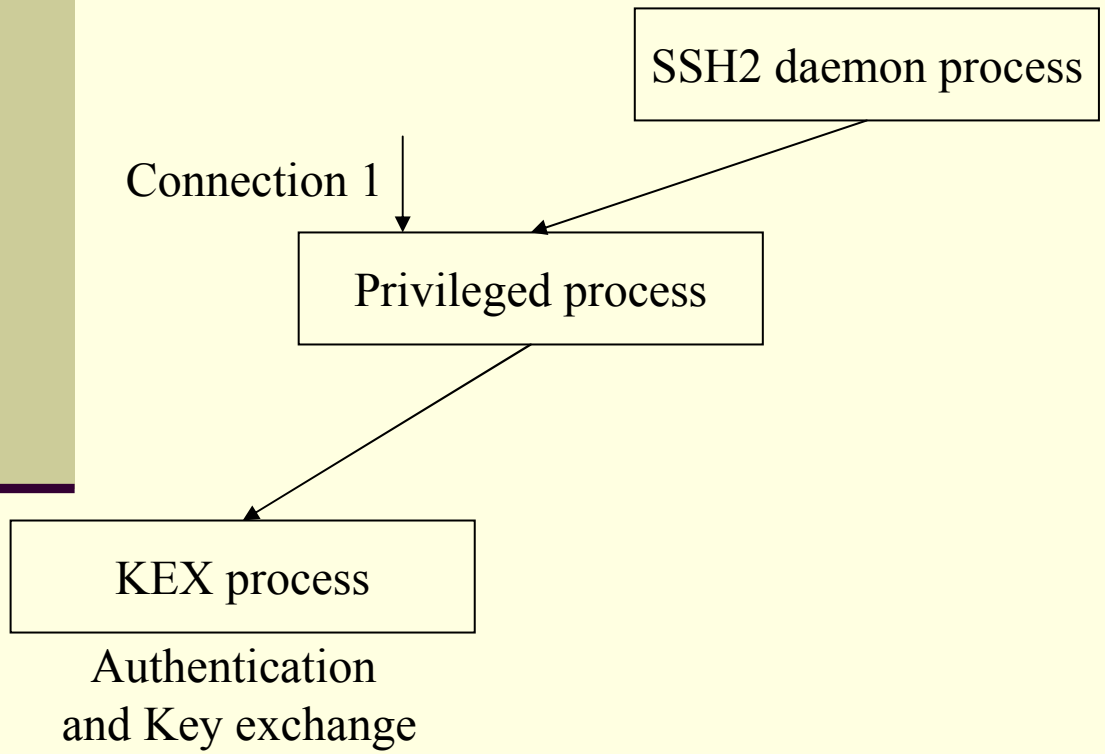
SSH process


Connection 1
SSH2 daemon process
Connection 1
Privileged process

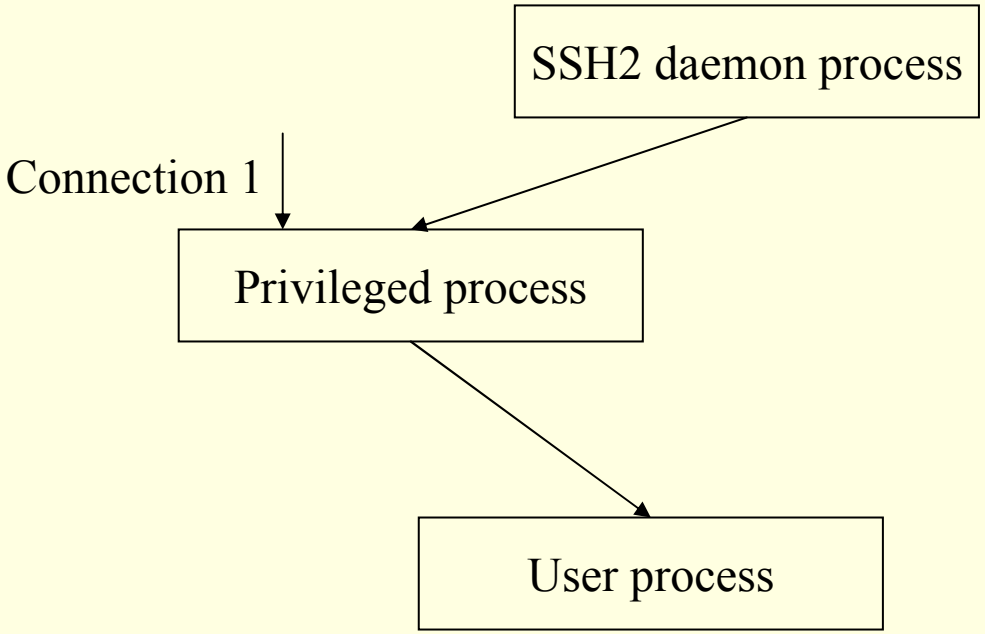
SSH process



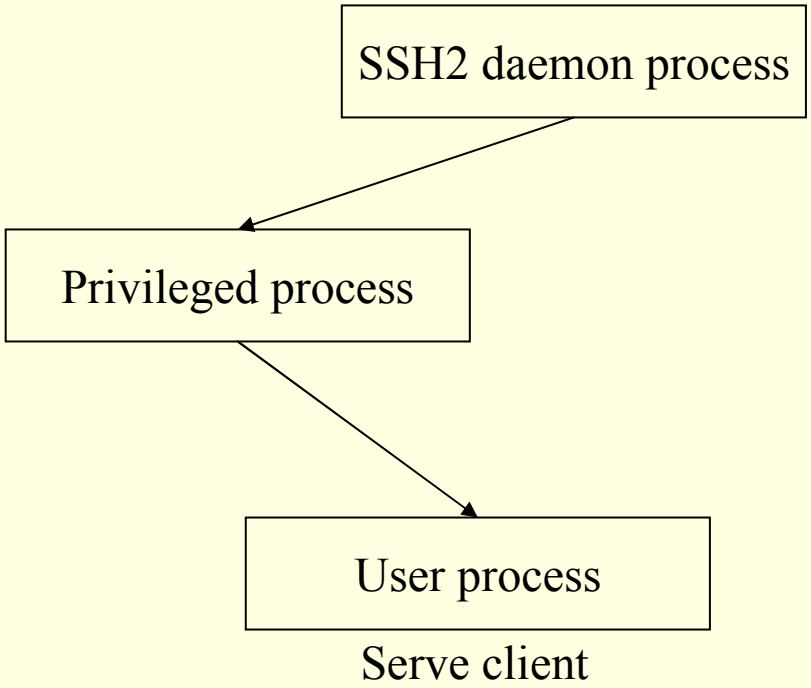
SSH process


SSH2 daemon process
Connection 1
Privileged process
KEX process
Authentication
and Key exchange

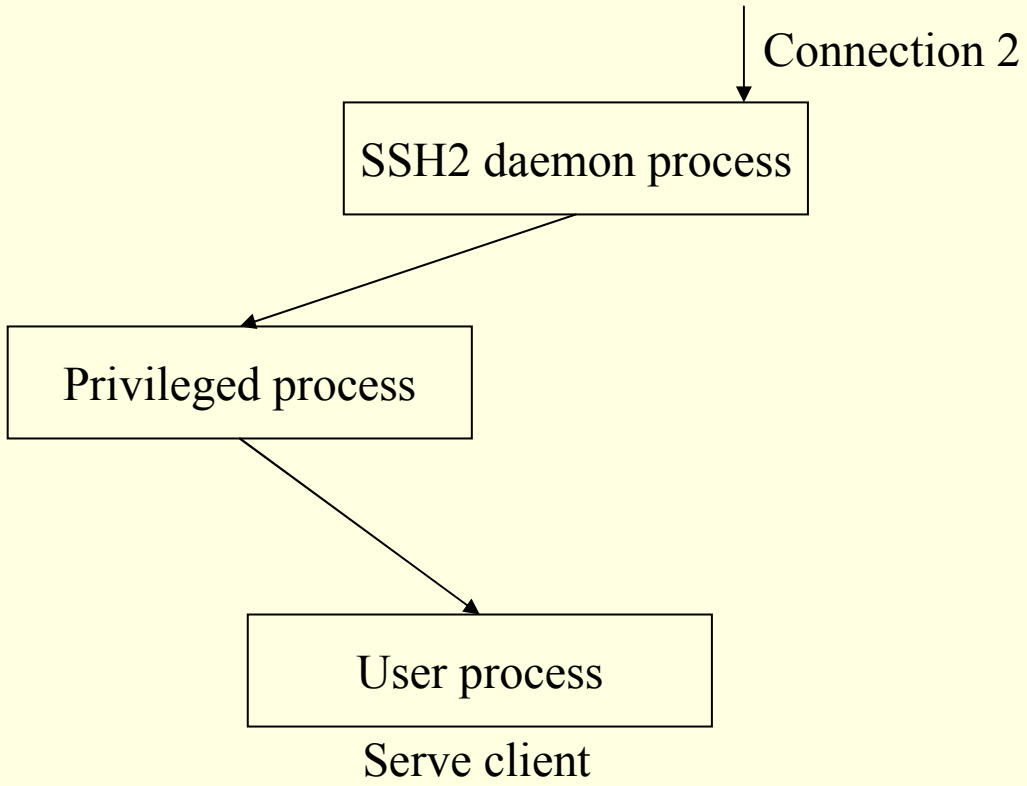
SSH process


SSH2 daemon process
Connection 1
Privileged process
User process

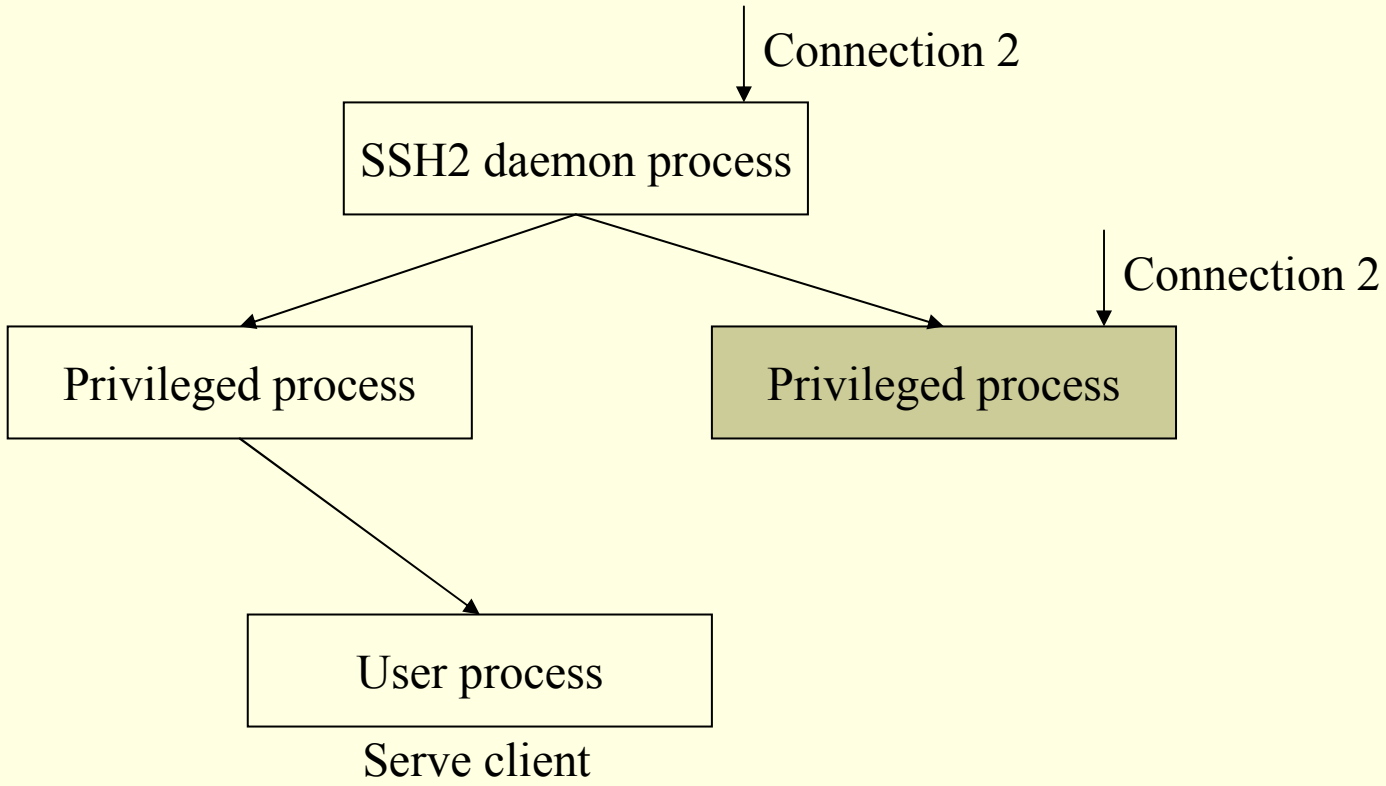
SSH process


SSH2 daemon process
Privileged process
User process
Serve client

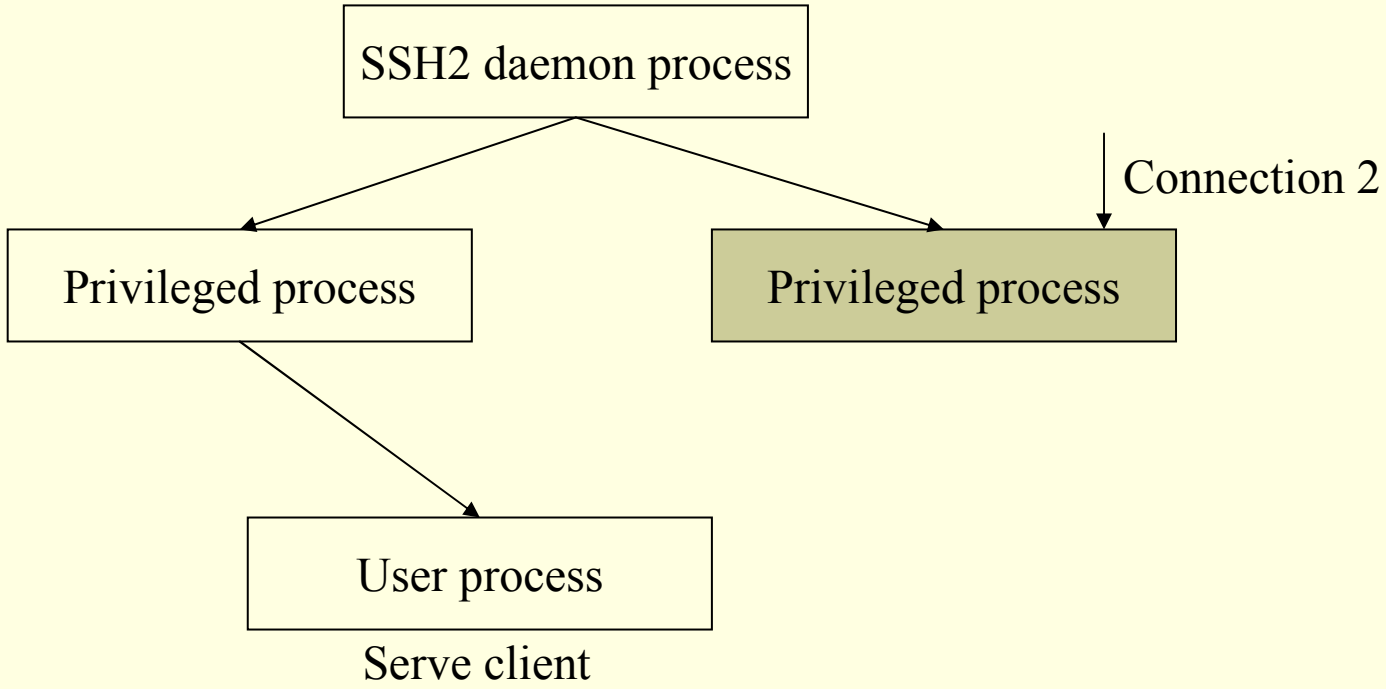
SSH process


Connection 2
SSH2 daemon process
Privileged process
User process
Serve client

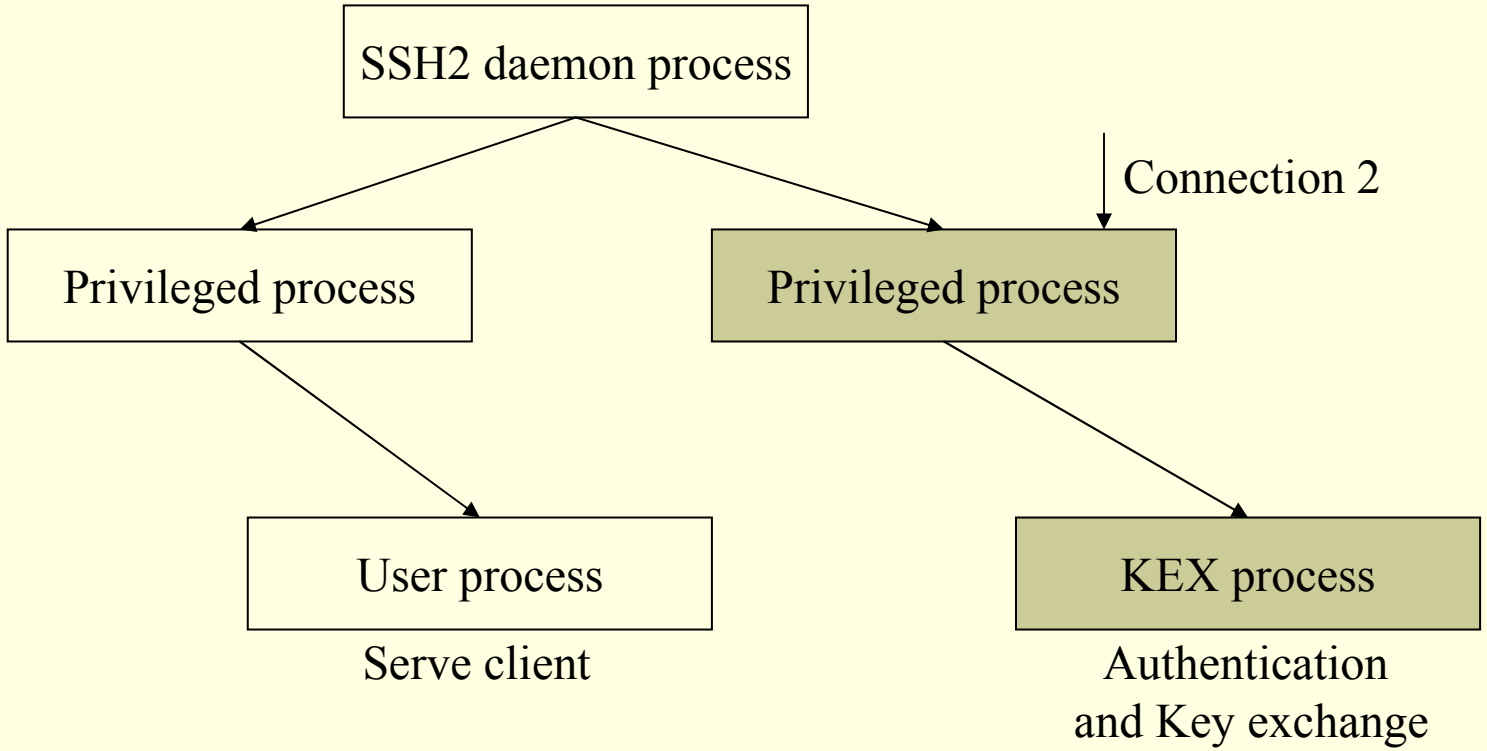
SSH process


Connection 2
SSH2 daemon process
Connection 2
Privileged process
Privileged process
User process
Serve client

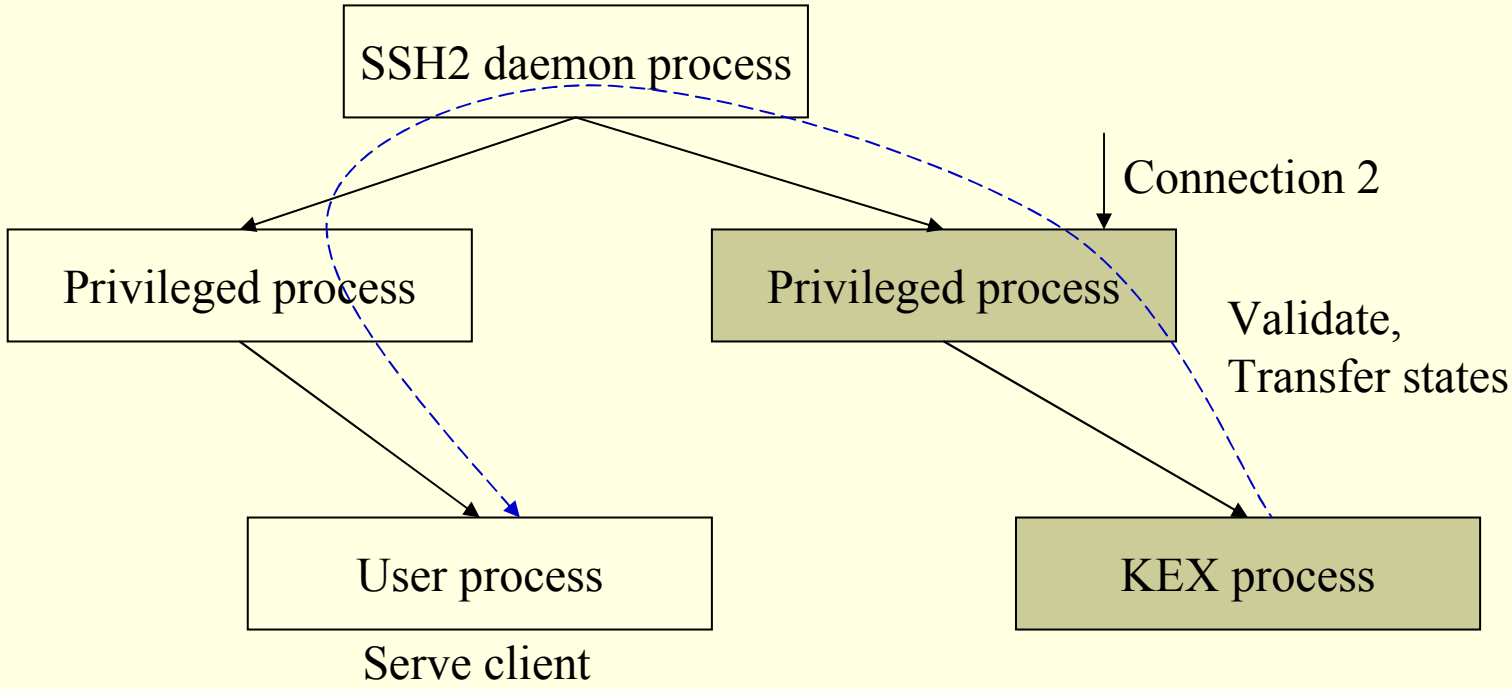
SSH process


SSH2 daemon process
Connection 2
Privileged process
Privileged process
User process
Serve client

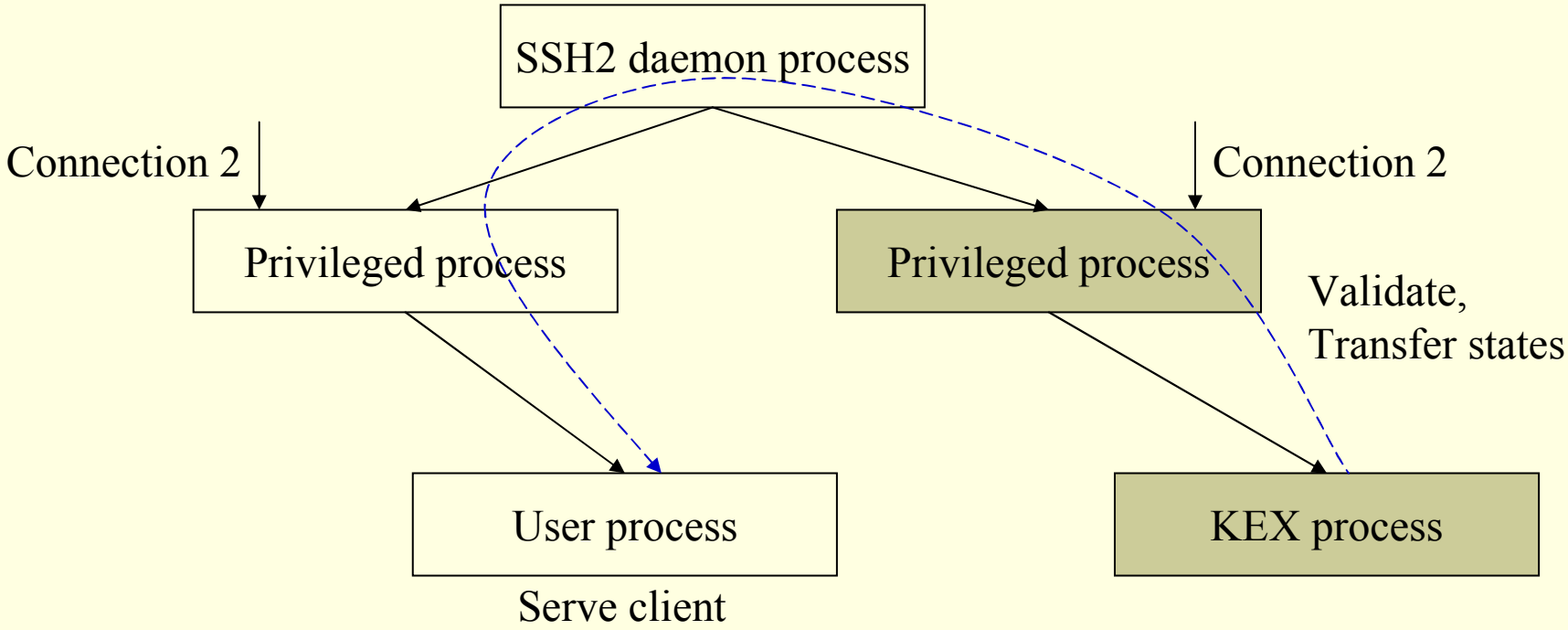
SSH process


SSH2 daemon process
Connection 2
Privileged process
Privileged process
User process
KEX process
Serve client
Authentication
and Key exchange

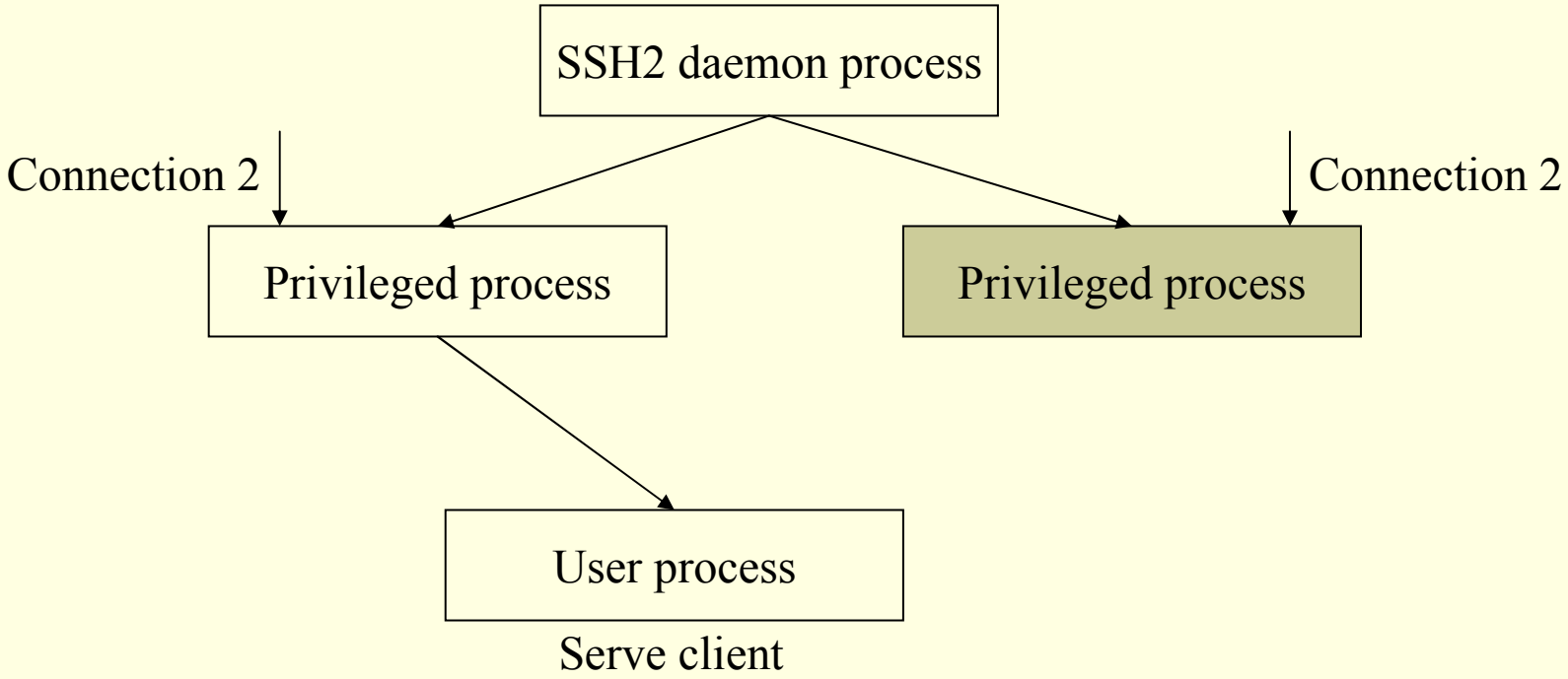
SSH process


SSH2 daemon process
Connection 2
Privileged process
Privileged process
Validate,
Transfer states
User process
KEX process
Serve client

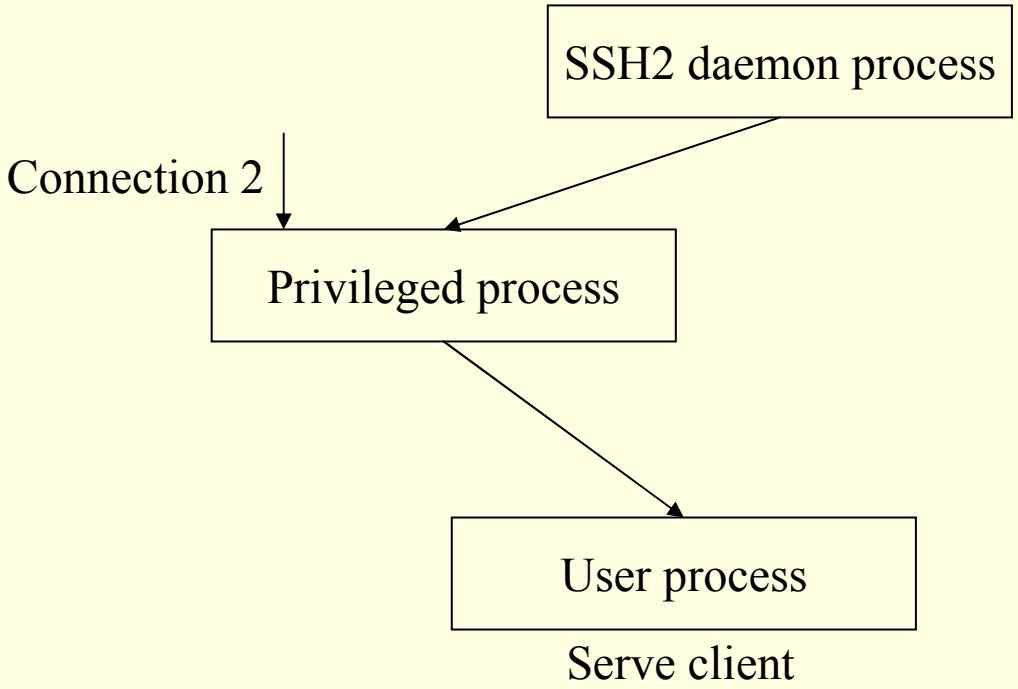
SSH process


SSH2 daemon process
Connection 2
Connection 2
Privileged process
Privileged process
Validate,
Transfer states
User process
KEX process
Serve client

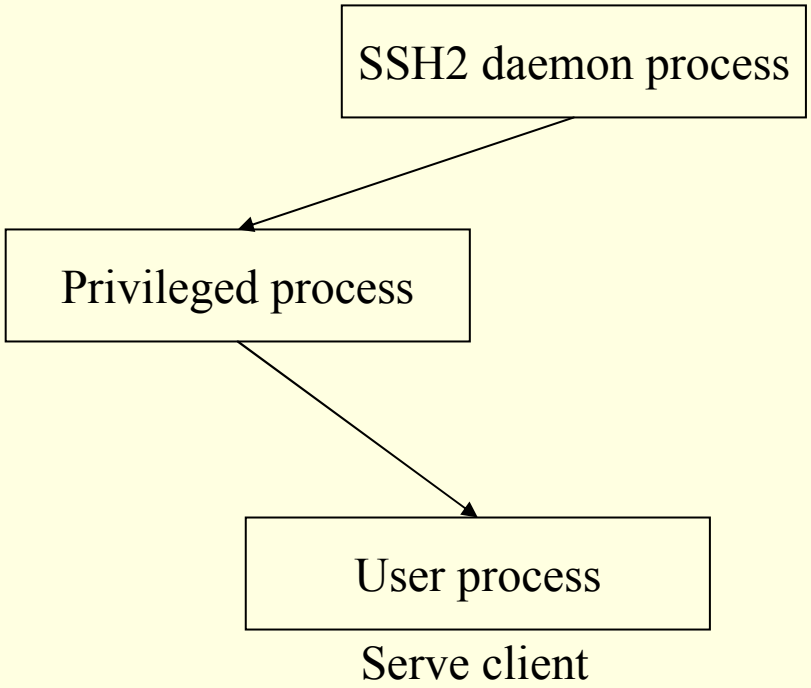
SSH process


SSH2 daemon process
Connection 2
Connection 2
Privileged process
Privileged process
User process
Serve client

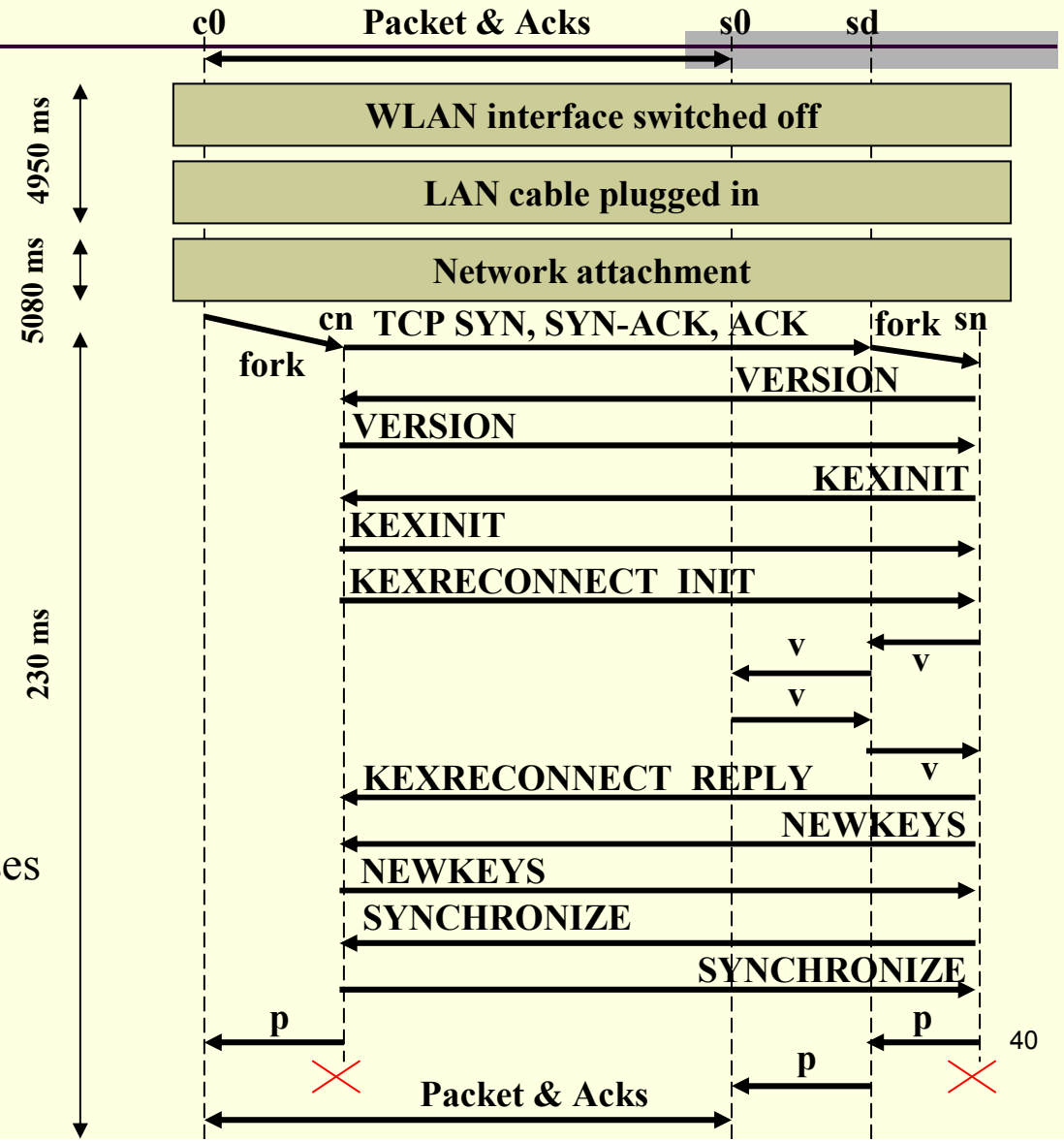
SSH process


SSH2 daemon process
Connection 2
Privileged process
User process
Serve client

SSH process


SSH2 daemon process
Privileged process
User process
Serve client

SSH process/ reconnecting session



'c0, s0': old client and server processes
'cn, sn': temporary processes
'sd': main daemon process
'v': inter-process messaging
'p': actual state passing

Outline

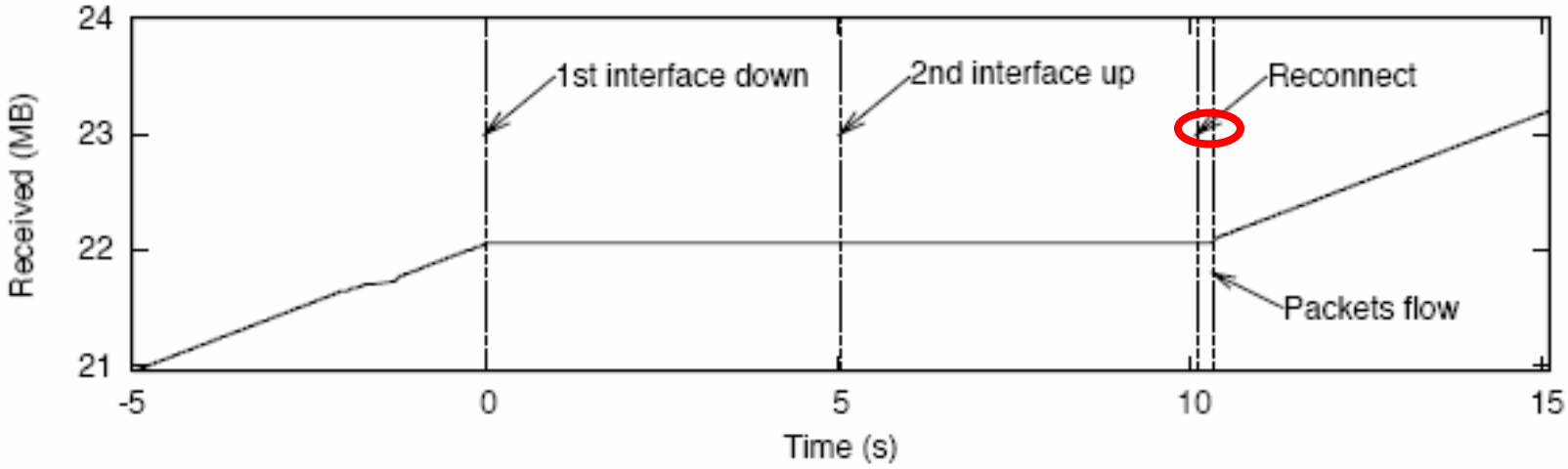
- Introduction
- Related work and Goals
- Design principles
- SSH/TLS extension procedure
- Implementation considerations
- **Evaluation**
- Conclusions

Scenario

- Manually switches from WLAN to wired Ethernet
  - A client network interface goes down, second comes up, client reconnects.
- Downloads a large file from a remote server
  - Over SFTP (FTP over SSH)
  - Over TLS (FTP over TLS)
- Metric: the actual expected length of typical reconnections

OpenSSH

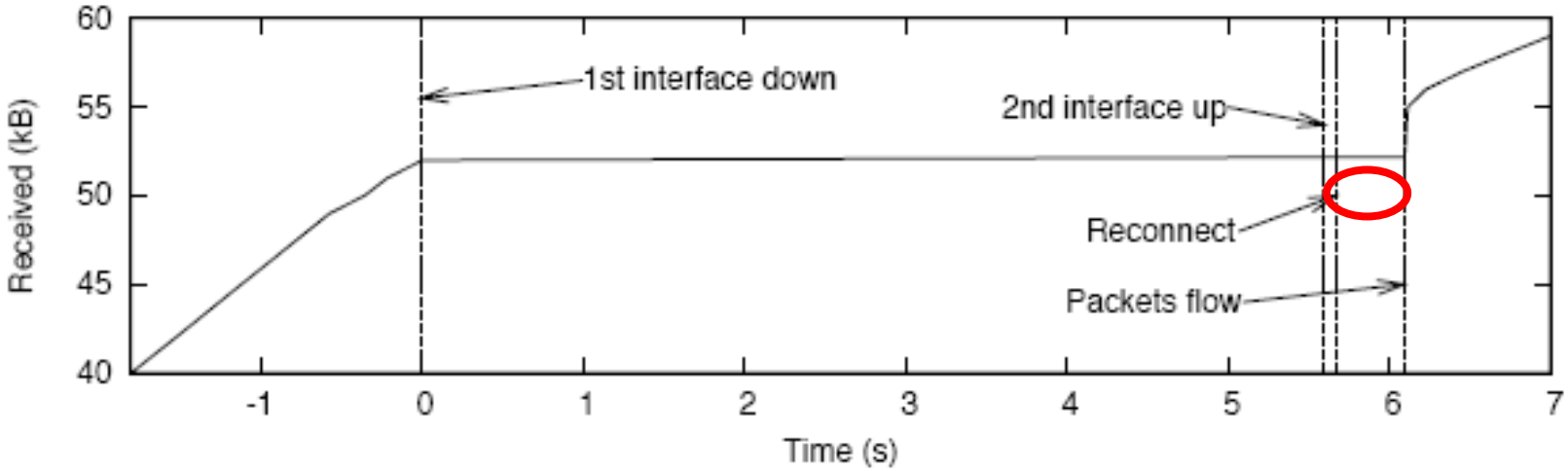
- 10ms RTT through WLAN an LAN, SFTP client runs on MacOSX 10.4, SSH server runs on Linux.
- 5sec: off WLAN, 5sec: up Ethernet, 0.2sec: reconnection



Progress of an SFTP transfer before, during, and after reconnection

PureTLS

- 1ms RTT, the client runs on Linux, the server runs on Windows XP
- ~5.5sec (off WLAN + up Ethernet), 0.5sec: reconnection



Progress of an TLS transfer before, during, and after reconnection

Overhead & complexity

- Acknowledgment overhead
  - No additional IP packets: SSH/TLS ACKs fits in the same TCP ACKs.
  - OpenSSH: Send ACKs every received SSH message
  - PureTLS: Send ACKs at the same time as application data
- Implementation complexity
  - Need modification at Server/Client applications
  - OpenSSH 2,200 lines of extension code
  - PureTLS 1,000 lines of extension code

Outline

- Introduction
- Related work and Goals
- Design principles
- SSH/TLS extension procedure
- Implementation considerations
- Evaluation
- **Conclusions**

Conclusion

- Extend SSH and TLS to support resilient connections (Handle long periods of disconnected operation)
- Some principles having an effect to deployability were identified and tested
- Deployability remains a difficult issue.

- Thank you! Question?

Issues & Questions here?

- Questions?
  - ACKs Overhead:
    - A single ACK is 32 bytes in total. Since the SSH transport layer messages can be up to 32 KB: the ACK traffic amounted to less than 0.6% of the whole bandwidth.?
  - Laptop Suspension: how can they save all states? Maybe no need since they don't care; just reconnect.
  - This work does not talk about how to operate the resilient when the laptop is suspended.

Issues & Questions here?

- Questions?
  - Also, they assume the old process is always up->find out how to operate when WINDOW is suspended)
  - Might be a mechanism to inform the client how long the server will hold the connection based on the local policy -> TLS library either to use this function or not (resilient = no but support)
  - No optimization, (At least, do analysis on reconnection part)