Semantics and Verification of Software

Winter Semester 2017/18

Lecture 19: Security Type Systems

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ws-1718/sv-sw/

Outline of Lecture 19

Recap: Type Systems

Typed Execution of Statements

Definition (Typed execution of statements (cf. Definition 16.4))

$\rightarrow_1 \subseteq (\mathit{Cmd} \times \Sigma) \times (\mathit{Cmd} \times \Sigma)$ is given by:

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma \rangle} \qquad \frac{\langle a, \sigma \rangle \rightarrow v}{\langle x := a, \sigma \rangle \rightarrow_1 \langle \downarrow, \sigma[x \mapsto v] \rangle}$$

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle} \qquad \frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow_1 \langle c_1'; c_2, \sigma' \rangle}$$

$$\ldots \qquad \ldots$$

Thus: execution gets stuck iff expression evaluation gets stuck

Example

For $c = (\texttt{x := x + 1})$,

- if $\sigma(\texttt{x}) = 2 \in \mathbb{Z}$, then $\langle c, \sigma \rangle \rightarrow_1 \sigma[\texttt{x} \mapsto 3]$
- if $\sigma(\texttt{x}) = 3.14 \in \mathbb{R}$, then $\langle c, \sigma \rangle \not\rightarrow_1$

Recap: Type Systems

Data Types and Type Environments

- Data types

$$Typ = \{\text{int}, \text{real}\}$$

  with $\tau \in Typ$
- Observation: type of (arithmetic) expression depends on types of contained variables
- Therefore: introduce type environments

$$TEnv = \{\Gamma \mid \Gamma : Var \to Typ\}$$

- In the following: typing rule systems for
  - arithmetic expression $a \in AExp$ has type $\tau \in Typ$: $\Gamma \vdash a : \tau$
  - Boolean expression $b \in BExp$ is well-typed: $\Gamma \vdash b$
  - statement $c \in Cmd$ is well-typed: $\Gamma \vdash c$

  in type environment $\Gamma \in TEnv$

Recap: Type Systems

Typing Rules for Commands

Definition (Well-typedness of commands)

For $\Gamma \in \mathit{TEnv}$ and $c \in \mathit{Cmd}$, $\Gamma \vdash c$ is given by

$$\frac{}{\Gamma \vdash \texttt{skip}} \qquad \frac{\Gamma \vdash a : \Gamma(x)}{\Gamma \vdash x := a} \qquad \frac{\Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash c_1 \,;\, c_2}$$

$$\frac{\Gamma \vdash b \quad \Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}} \qquad \frac{\Gamma \vdash b \quad \Gamma \vdash c}{\Gamma \vdash \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}}$$

Thus: command not well-typed iff some expression not well-typed or assignment incompatible

Recap: Type Systems

Type Soundness for Commands

Theorem (Progress)

If $c \in Cmd \setminus \{\downarrow\}$, $\sigma \in \Sigma$ *and* $\Gamma \in TEnv$ *such that* $\Gamma \vdash c$ *and* $\Gamma \vdash \sigma$, *then there exist* $c' \in Cmd$ *and* $\sigma' \in \Sigma$ *such that* $\langle c, \sigma \rangle \to_1 \langle c', \sigma' \rangle$.

Theorem (Preservation)

If $c, c' \in Cmd$, $\sigma, \sigma' \in \Sigma$ *and* $\Gamma \in TEnv$ *such that* $\Gamma \vdash c$, $\Gamma \vdash \sigma$ *and* $\langle c, \sigma \rangle \to_1 \langle c', \sigma' \rangle$, *then* $\Gamma \vdash c'$ *and* $\Gamma \vdash \sigma'$.

Corollary (Type soundness)

If $c, c' \in Cmd$, $\sigma, \sigma' \in \Sigma$ *and* $\Gamma \in TEnv$ *such that* $\Gamma \vdash c$, $\Gamma \vdash \sigma$ *and* $\langle c, \sigma \rangle \to_1^* \langle c', \sigma' \rangle$ *with* $c' \neq \downarrow$, *then there exist* $c'' \in Cmd$ *and* $\sigma'' \in \Sigma$ *such that* $\langle c', \sigma' \rangle \to_1 \langle c'', \sigma'' \rangle$.

All proofs by rule induction (based on the corresponding results for expressions)

Outline of Lecture 19

Language-Based Security

Information flow control

Goal: Ensure that programs protect private data like passwords, bank details, or medical records. There should be no information flow from private data into public channels.

Language-Based Security

Information flow control

Goal: Ensure that programs protect private data like passwords, bank details, or medical records. There should be no information flow from private data into public channels.

Language-based security

Approach to information flow control where data flow analysis is used to determine whether a program is free of illicit information flows

- guarantees confidentiality by program analysis, not by cryptography
- analyses often expressed as type systems

Security Levels

- Data objects (such as program variables) have security/confidentiality levels
- Security levels are partially ordered:

  $l < l'$ means that l is less confidential than l'

  (thus, information is allowed to flow from l to l' but not vice versa)
- Simplest choice: low (low/public) and high (high/confidential/secret/private) with low $<$ high
- Extension to multi-level security by generalisation to lattice SL
- Security levels of variables given by $\mathsf{sl} : Var \to SL$

Security Levels

- Data objects (such as program variables) have security/confidentiality levels
- Security levels are partially ordered:

  $l < l'$ means that l is less confidential than l'

  (thus, information is allowed to flow from l to l' but not vice versa)
- Simplest choice: low (low/public) and high (high/confidential/secret/private) with low $<$ high
- Extension to multi-level security by generalisation to lattice SL
- Security levels of variables given by $\mathsf{sl} : \mathit{Var} \to \mathit{SL}$

Example 19.1 (Illicit flows)

Let $\mathsf{sl}(\texttt{low}) = \mathsf{low}$ and $\mathsf{sl}(\texttt{high}) = \mathsf{high}$.

Explicit: `low := high`

Implicit ("occlusion"): `if high = 1 then low := 1 else low := 2 end`

Security Levels on Expressions

The security level of an expression is the maximal security level of any of its variables.

Definition 19.2 (Security levels on expressions)

$\mathsf{sl} : \mathit{Var} \to \mathit{SL}$ is extended to $\mathsf{sl} : \mathit{AExp} \cup \mathit{BExp} \to \mathit{SL}$ as follows:

$$\begin{aligned} \mathsf{sl}(z) &= \min \mathit{SL} \\ \mathsf{sl}(a_1 + a_2) &= \mathsf{sl}(a_1) \sqcup \mathsf{sl}(a_2) \\ &\dots \end{aligned} \qquad \begin{aligned} \mathsf{sl}(t) &= \min \mathit{SL} \\ \mathsf{sl}(a_1 = a_2) &= \mathsf{sl}(a_1) \sqcup \mathsf{sl}(a_2) \\ \mathsf{sl}(\neg b) &= \mathsf{sl}(b) \\ \mathsf{sl}(b_1 \wedge b_2) &= \mathsf{sl}(b_1) \sqcup \mathsf{sl}(b_2) \\ &\dots \end{aligned}$$

Security Levels on States

Idea: identify when states can (not) be distinguished by a “low observer”

Definition 19.3 (Security levels on states)

Two program states $\sigma_1, \sigma_2 \in \Sigma$ agree up to level $l \in SL$ if for all $x \in Var$ such that $\mathsf{sl}(x) \leq l$,

$$\sigma_1(x) = \sigma_2(x).$$

Notation: $\sigma_1 =_{\leq l} \sigma_2$ (analogously: $\sigma_1 =_{< l} \sigma_2$)

Non-Interference Informally

To guarantee confidentiality, high variables must not interfere with low ones, i.e., a variation of confidential input must not cause a variation of public output.

Non-Interference Informally

To guarantee confidentiality, high variables must not interfere with low ones, i.e., a variation of confidential input must not cause a variation of public output.

Non-interference (informally)

Program c guarantees non-interference if for all states σ_1, σ_2: If σ_1 and σ_2 agree on low variables (but possibly differ on high variables), then the states resulting from executing $\langle c, \sigma_1 \rangle$ and $\langle c, \sigma_2 \rangle$ must also agree on low variables.

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \mathit{Cmd}$ guarantees non-interference at security level $l \in \mathit{SL}$ if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \to \sigma_1', \langle c, \sigma_2 \rangle \to \sigma_2' \Rightarrow \sigma_1' =_{\leq l} \sigma_2'$$

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \mathit{Cmd}$ guarantees non-interference at security level $l \in \mathit{SL}$ if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \to \sigma_1', \langle c, \sigma_2 \rangle \to \sigma_2' \Rightarrow \sigma_1' =_{\leq l} \sigma_2'$$

Thus: every program c guarantees non-interference at level $l_c = \bigsqcup \{ \mathsf{sl}(x) \mid x \in \mathit{Var}_c \}$

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \mathit{Cmd}$ guarantees non-interference at security level $l \in \mathit{SL}$ if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \to \sigma_1', \langle c, \sigma_2 \rangle \to \sigma_2' \Rightarrow \sigma_1' =_{\leq l} \sigma_2'$$

Thus: every program c guarantees non-interference at level $l_c = \bigsqcup \{ \mathsf{sl}(x) \mid x \in \mathit{Var}_c \}$

Example 19.5 (Interference (cf. Example 19.1))

1. $c_1 = (\texttt{low := high})$ does not guarantee non-interference at level low:
   - let $\sigma_i = [\texttt{low} \mapsto 0, \texttt{high} \mapsto i]$ for $i = 1, 2$ (and thus $\sigma_1 =_{\leq \mathsf{low}} \sigma_2$)
   - then $\langle c_1, \sigma_1 \rangle \to \sigma_1'$ and $\langle c_1, \sigma_2 \rangle \to \sigma_2'$ with $\sigma_i'(\texttt{low}) = i$ $(i = 1, 2)$
   - thus $\sigma_1' \neq_{\leq \mathsf{low}} \sigma_2'$.

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \mathit{Cmd}$ guarantees non-interference at security level $l \in \mathit{SL}$ if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \to \sigma_1', \langle c, \sigma_2 \rangle \to \sigma_2' \Rightarrow \sigma_1' =_{\leq l} \sigma_2'$$

Thus: every program c guarantees non-interference at level $l_c = \bigsqcup \{ \mathsf{sl}(x) \mid x \in \mathit{Var}_c \}$

Example 19.5 (Interference (cf. Example 19.1))

1. $c_1 = (\texttt{low := high})$ does not guarantee non-interference at level low:
   - let $\sigma_i = [\texttt{low} \mapsto 0, \texttt{high} \mapsto i]$ for $i = 1, 2$ (and thus $\sigma_1 =_{\leq \text{low}} \sigma_2$)
   - then $\langle c_1, \sigma_1 \rangle \to \sigma_1'$ and $\langle c_1, \sigma_2 \rangle \to \sigma_2'$ with $\sigma_i'(\texttt{low}) = i$ $(i = 1, 2)$
   - thus $\sigma_1' \neq_{\leq \text{low}} \sigma_2'$.
2. Similarly: $c_2 = (\texttt{if high = 1 then low := 1 else low := 2 end})$ does not guarantee non-interference at level low

Non-interference Formally

Definition 19.4 (Non-interference)

Program $c \in \mathit{Cmd}$ guarantees non-interference at security level $l \in \mathit{SL}$ if for all $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$:

$$\sigma_1 =_{\leq l} \sigma_2, \langle c, \sigma_1 \rangle \to \sigma_1', \langle c, \sigma_2 \rangle \to \sigma_2' \Rightarrow \sigma_1' =_{\leq l} \sigma_2'$$

Thus: every program c guarantees non-interference at level $l_c = \bigsqcup \{ \mathsf{sl}(x) \mid x \in \mathit{Var}_c \}$

Example 19.5 (Interference (cf. Example 19.1))

1. $c_1 = (\texttt{low := high})$ does not guarantee non-interference at level low:
   - let $\sigma_i = [\texttt{low} \mapsto 0, \texttt{high} \mapsto i]$ for $i = 1, 2$ (and thus $\sigma_1 =_{\leq \text{low}} \sigma_2$)
   - then $\langle c_1, \sigma_1 \rangle \to \sigma_1'$ and $\langle c_1, \sigma_2 \rangle \to \sigma_2'$ with $\sigma_i'(\texttt{low}) = i$ $(i = 1, 2)$
   - thus $\sigma_1' \neq_{\leq \text{low}} \sigma_2'$.
2. Similarly: $c_2 = (\texttt{if high = 1 then low := 1 else low := 2 end})$ does not guarantee non-interference at level low
3. Both programs guarantee non-interference at level high

Outline of Lecture 19

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees non-interference for well-typed programs

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees non-interference for well-typed programs
- Explicit flows easy to handle: for each assignment $x := a$, check that $\mathsf{sl}(x) \geq \mathsf{sl}(a)$

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees non-interference for well-typed programs
- Explicit flows easy to handle: for each assignment $x := a$, check that $\mathsf{sl}(x) \geq \mathsf{sl}(a)$
- Approach to deal with implicit flows: parametrise typing rules with security level of Boolean expressions ("`if high = 1`") that guard the current command ("`then low := 1`")

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees non-interference for well-typed programs
- Explicit flows easy to handle: for each assignment $x := a$, check that $\mathrm{sl}(x) \geq \mathrm{sl}(a)$
- Approach to deal with implicit flows: parametrise typing rules with security level of Boolean expressions ("`if high = 1`") that guard the current command ("`then low := 1`")
- Well-typedness predicate for $l \in SL$ and $c \in Cmd$:

$$l \vdash c$$

Meaning: "In the context of Boolean expressions of level (at most) l, command c is well-typed."

Handling Explicit and Implicit Flows

- **Goal:** develop type system that guarantees non-interference for well-typed programs
- Explicit flows easy to handle: for each assignment $x := a$, check that $\mathrm{sl}(x) \geq \mathrm{sl}(a)$
- Approach to deal with implicit flows: parametrise typing rules with security level of Boolean expressions ("`if high = 1`") that guard the current command ("`then low := 1`")
- Well-typedness predicate for $l \in SL$ and $c \in Cmd$:

$$l \vdash c$$

  Meaning: "In the context of Boolean expressions of level (at most) l, command c is well-typed."
- Hence: assignments to variables of level $< l$ are forbidden in c

The Security Type System

Definition 19.6 (Well-typedness of commands)

For $l \in SL$ and $c \in Cmd$, $l \vdash c$ is given by

$$\text{(s-skip)}\ \frac{}{l \vdash \texttt{skip}} \qquad \text{(s-asgn)}\ \frac{\mathsf{sl}(x) \geq \mathsf{sl}(a) \quad \mathsf{sl}(x) \geq l}{l \vdash x := a} \qquad \text{(s-seq)}\ \frac{l \vdash c_1 \quad l \vdash c_2}{l \vdash c_1 ; c_2}$$

$$\text{(s-if)}\ \frac{l \sqcup \mathsf{sl}(b) \vdash c_1 \quad l \sqcup \mathsf{sl}(b) \vdash c_2}{l \vdash \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}} \qquad \text{(s-wh)}\ \frac{l \sqcup \mathsf{sl}(b) \vdash c}{l \vdash \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}}$$

The Security Type System

Definition 19.6 (Well-typedness of commands)

For $l \in SL$ and $c \in Cmd$, $l \vdash c$ is given by

$$\text{(s-skip)}\ \frac{}{l \vdash \texttt{skip}} \qquad \text{(s-asgn)}\ \frac{\mathsf{sl}(x) \geq \mathsf{sl}(a) \quad \mathsf{sl}(x) \geq l}{l \vdash x := a} \qquad \text{(s-seq)}\ \frac{l \vdash c_1 \quad l \vdash c_2}{l \vdash c_1 ; c_2}$$

$$\text{(s-if)}\ \frac{l \sqcup \mathsf{sl}(b) \vdash c_1 \quad l \sqcup \mathsf{sl}(b) \vdash c_2}{l \vdash \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}} \qquad \text{(s-wh)}\ \frac{l \sqcup \mathsf{sl}(b) \vdash c}{l \vdash \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}}$$

Example 19.7

Let $SL = \{\mathsf{low}, \mathsf{high}, \mathsf{secret}\}$ (on the board) with $\mathsf{low} < \mathsf{high} < \mathsf{secret}$.

1. $\mathsf{low} \vdash$ `if high = low then high := 1 else skip end`
2. $\mathsf{low} \nvdash$ `if high = low then low := 1 else skip end`
3. $\mathsf{high} \vdash$ `if high = low then high := 1 else skip end`
4. $\mathsf{secret} \nvdash$ `if high = low then high := 1 else skip end`

Confinement

The following lemma states that if $l \vdash c$, then c cannot modify variables of level $< l$, i.e., the effect of c is confined to variables of level $\geq l$:

Lemma 19.8 (Confinement)

Let $c \in Cmd$ and $l \in SL$ such that $l \vdash c$. Then for all $\sigma, \sigma' \in \Sigma$ with $\langle c, \sigma \rangle \to \sigma'$,

$$\sigma =_{<l} \sigma'$$

Proof.

omitted □

Soundness

The following theorem states that the security type system is sound, i.e., that well-typedness guarantees non-interference at all levels:

Theorem 19.9 (Non-interference)

Let $c \in Cmd$ such that $\min SL \vdash c$, and let $l \in SL$ and $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$ such that $\sigma_1 =_{\leq l} \sigma_2$, $\langle c, \sigma_1 \rangle \to \sigma_1'$, and $\langle c, \sigma_2 \rangle \to \sigma_2'$. Then $\sigma_1' =_{\leq l} \sigma_2'$.

Soundness

The following theorem states that the security type system is sound, i.e., that well-typedness guarantees non-interference at all levels:

Theorem 19.9 (Non-interference)

Let $c \in Cmd$ such that $\min SL \vdash c$, and let $l \in SL$ and $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$ such that $\sigma_1 =_{\leq l} \sigma_2$, $\langle c, \sigma_1 \rangle \to \sigma_1'$, and $\langle c, \sigma_2 \rangle \to \sigma_2'$. Then $\sigma_1' =_{\leq l} \sigma_2'$.

Proof.

omitted □

Soundness

The following theorem states that the security type system is sound, i.e., that well-typedness guarantees non-interference at all levels:

Theorem 19.9 (Non-interference)

Let $c \in Cmd$ such that $\min SL \vdash c$, and let $l \in SL$ and $\sigma_1, \sigma_2, \sigma_1', \sigma_2' \in \Sigma$ such that $\sigma_1 =_{\leq l} \sigma_2$, $\langle c, \sigma_1 \rangle \to \sigma_1'$, and $\langle c, \sigma_2 \rangle \to \sigma_2'$. Then $\sigma_1' =_{\leq l} \sigma_2'$.

Proof.

omitted □

The following example shows that the type system is incomplete, i.e., rejects some programs that guarantee non-interference:

Example 19.10

$\text{low} \nvdash$ `if high = 1 then low := 1 else low := 1 end`

Extensions

- Does non-interference really guarantee absence of information flow?
  - problem: covert channels
  - example: $c = \texttt{while high = 1 do skip end}$
  - then $\min SL \vdash c$
  - but non-termination of c allows to conclude that $\texttt{high = 1}$
- Other kinds of covert channels: processing time, power consumption, ...

Extensions

- Does non-interference really guarantee absence of information flow?
  - problem: covert channels
  - example: $c = \texttt{while high = 1 do skip end}$
  - then $\min SL \vdash c$
  - but non-termination of c allows to conclude that $\texttt{high = 1}$
- Other kinds of covert channels: processing time, power consumption, ...
- Encryption breaks traditional non-interference as (public) ciphertexts do depend on (confidential) contents
  - example: $low := encrypt(high)$
  - then $\sigma_1(\texttt{high}) \neq \sigma_2(\texttt{high})$ (usually) implies $\sigma_1'(\texttt{low}) \neq \sigma_2'(\texttt{low})$ after assignment
  - thus $\sigma_1' \not\approx_{\leq \text{low}} \sigma_2'$ (interference)

Extensions

- Does non-interference really guarantee absence of information flow?
  - problem: covert channels
  - example: $c = \texttt{while high = 1 do skip end}$
  - then $\min SL \vdash c$
  - but non-termination of c allows to conclude that $\texttt{high = 1}$
- Other kinds of covert channels: processing time, power consumption, ...
- Encryption breaks traditional non-interference as (public) ciphertexts do depend on (confidential) contents
  - example: $low := encrypt(high)$
  - then $\sigma_1(\texttt{high}) \neq \sigma_2(\texttt{high})$ (usually) implies $\sigma_1'(\texttt{low}) \neq \sigma_2'(\texttt{low})$ after assignment
  - thus $\sigma_1' \not\approx_{\leq \text{low}} \sigma_2'$ (interference)
- Solution: relaxed notions of (non-)interference that distinguish between
  - breaking non-interference because of legitimate use of (sufficiently strong) encryption and
  - breaking non-interference due to an (unintended) leak

  ("possibilistic non-interference")

Extensions

- Does non-interference really guarantee absence of information flow?
  - problem: covert channels
  - example: $c = \texttt{while high = 1 do skip end}$
  - then $\min SL \vdash c$
  - but non-termination of c allows to conclude that $\texttt{high = 1}$
- Other kinds of covert channels: processing time, power consumption, ...
- Encryption breaks traditional non-interference as (public) ciphertexts do depend on (confidential) contents
  - example: $low := encrypt(high)$
  - then $\sigma_1(\texttt{high}) \neq \sigma_2(\texttt{high})$ (usually) implies $\sigma_1'(\texttt{low}) \neq \sigma_2'(\texttt{low})$ after assignment
  - thus $\sigma_1' \not\approx_{\leq \text{low}} \sigma_2'$ (interference)
- Solution: relaxed notions of (non-)interference that distinguish between
  - breaking non-interference because of legitimate use of (sufficiently strong) encryption and
  - breaking non-interference due to an (unintended) leak

  ("possibilistic non-interference")
- Excellent survey paper:

  A. Sabelfeld, A.C. Myers: Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications, 21(1), 2003, 5–19